PHY1610H - Scientific Computing: Partial Differential Equations

Ramses van Zon and Marcelo Ponce

SciNet HPC Consortium/Physics Department
University of Toronto

February, 2021

Today's class

Today we will discuss the following topics:

- Basic approaches to solving PDEs.
- How to approach the temporal part of the equations.
- How to approach the spatial part of the equations.

Numerical Methods

- **Finite Differences** $\rightsquigarrow$ (eg. matrix representation –*stencil*–, stability analysis, eigenvalues,...) $\Rightarrow$ Lineal Algebra
- **Spectral Methods** $\Rightarrow$ FFT
- **Implict Methods** $\Rightarrow$ Root Finding
- **Integration** $\Rightarrow$ Random Numbers & Monte Carlo
- Special Functions (eg. Green functions)

SciNet Physics UNIVERSITY OF TORONTO

Partial Differential Equations

Partial differential equations (PDEs) are differential equations which contain derivatives of more than one variable.

$$A\frac{\partial^2\Phi}{\partial x^2} + B\frac{\partial^2\Phi}{\partial x\partial y} + C\frac{\partial^2\Phi}{\partial y^2} = G\left(x, y, \Phi, \frac{\partial\Phi}{\partial x}, \frac{\partial\Phi}{\partial y}\right)$$

For A, B, C constant, three classes of PDEs show up repeatedly in physical systems.

- $B^2 - 4AC < 0$ – *elliptic* (Laplace's, Poisson equations).
- $B^2 - 4AC = 0$ – *parabolic* (Diffusion equation, Navier-Stokes).
- $B^2 - 4AC > 0$ – *hyperbolic* (Wave equation).

If there are packages that can solve PDEs in your field, then explore using them. However, writing your own is sometimes not a bad option.

SciNet Physics UNIVERSITY OF TORONTO

How do we solve these problems?

Let's take a look at a parabolic equation, in particular the heat equation (diffusion equation).

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2},$$

where T is the temperature and k is the thermal diffusivity.

How do we solve this equation? By discretizing in both space and time, and marching an initial condition forward in time.

This is similar to the initial-value problem in the ODEs, but in this case we have another derivative to deal with.

Dealing with time

If we rewrite the right-hand side as an operator:

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2} \quad \rightarrow \quad \frac{\partial T}{\partial t} = FT$$

we have two basic approaches to dealing with the time part of the equation:

- Explicit methods, such as forward Euler:

$$\frac{T_{i+1} - T_i}{\Delta t} = FT_i \quad \rightarrow \quad T_{i+1} = (1 + \Delta t F)T_i$$

- Implicit methods, such as backward Euler:

$$\frac{T_{i+1} - T_i}{\Delta t} = FT_{i+1} \quad \rightarrow \quad (1 - \Delta t F)T_{i+1} = T_i$$

Explicit Methods

Methods are called explicit when only $\boldsymbol{T_i}$ is on the right side of the equation. Explicit methods have some nice features:

- They are easy to implement.
- They are usually quick to calculate (no matrix inversions).
- Easier to parallelize, since the calculation is inherently local.
- There exist more-accurate explicit methods than forward Euler (Runge-Kutta, for example).

$$\frac{\boldsymbol{T_{i+1}} - \boldsymbol{T_i}}{\boldsymbol{\Delta t}} = \boldsymbol{F T_i} \quad \rightarrow \quad \boldsymbol{T_{i+1}} = (1 + \boldsymbol{\Delta t F})\boldsymbol{T_i}$$

But there are some serious downsides as well:

- They are not very accurate at low order ($\mathcal{O}(\boldsymbol{\Delta t})$ for forward Euler).
- They can be numerically unstable (though there are exceptions).

Implicit methods

Methods are called implicit when $\boldsymbol{T_{i+1}}$ is on both sides of the equation. Examples include backward Euler:

$$\frac{\boldsymbol{T_{i+1} - T_i}}{\boldsymbol{\Delta t}} = \boldsymbol{F T_{i+1}} \qquad \boldsymbol{\mathcal{O}(\Delta t)}$$

and Crank-Nicolson:

$$\frac{\boldsymbol{T_{i+1} - T_i}}{\boldsymbol{\Delta t}} = \boldsymbol{(F T_{i+1} + F T_i)/2} \qquad \boldsymbol{\mathcal{O}(\Delta t^2)}$$

Implicit time stepping methods have some nice features:

- They are stable over a wide range of timestep sizes, sometimes unconditionally (not unconditionally accurate, though).
- Excellent for solving steady-state problems.

Downsides include being more difficult to code, and much more difficult to parallelize (inverting the operator depends upon all grid points).

Dealing with space

We've talked about time. Now, what about space? How can we deal with the spatial aspects of the problem? How should set up our spatial discretization?

There are many different approaches one can take when dealing with fields.

- Finite difference methods.
- Finite volume methods.
- Finite element methods.
- Spectral methods.

And combinations thereof. How you set up your discretization will determine the structure of your $\boldsymbol{F}$ operator.

Finite difference methods

Finite difference methods are the simplest way to deal with your equations.

- The simulation domain is discretized.
- Derivatives are approximated by linear combinations of function values at the grid points.

We've used 'central differences' before, to calculate our second derivatives. Where did that come from?

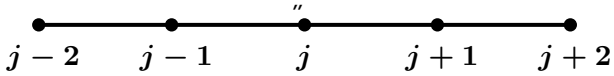
$$\frac{\partial^2 T_j}{\partial x^2} = \frac{T_{j+1} - 2T_j + T_{j-1}}{\Delta x^2}$$

Where j is the index of the x grid, and Δx is the spacing in x.

SciNet Physics UNIVERSITY OF TORONTO

Calculating derivatives

We need to calculate the second spatial derivatives. How best to do that? We discretize the x domain, and examine the Taylor expansion of some function f, centered around three different points:



$$
\begin{aligned}
f(x_{j-1}) &= f(x_j) - (\Delta x)\frac{\partial f(x_j)}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 f(x_j)}{\partial x^2} + \mathcal{O}(\Delta x^3) \\
f(x_j) &= f(x_j) \\
f(x_{j+1}) &= f(x_j) + (\Delta x)\frac{\partial f(x_j)}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 f(x_j)}{\partial x^2} + \mathcal{O}(\Delta x^3)
\end{aligned}
$$

Where $\Delta x = x_j - x_{j-1}$.

SciNet Physics UNIVERSITY OF TORONTO

Calculating derivatives, continued

We can write this as a matrix operation:

$$\begin{bmatrix} f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta x & \frac{\Delta x^2}{2} \\ 1 & 0 & 0 \\ 1 & \Delta x & \frac{\Delta x^2}{2} \end{bmatrix} \begin{bmatrix} f(x_j) \\ f'(x_j) \\ f''(x_j) \end{bmatrix}$$

To get the answer we invert the matrix:

$$\begin{bmatrix} f(x_j) \\ f'(x_j) \\ f''(x_j) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{-1}{2\Delta x} & 0 & \frac{1}{2\Delta x} \\ \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} \end{bmatrix} \begin{bmatrix} f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \end{bmatrix}$$

$$f'(x_j) = \frac{\partial f(x_j)}{\partial x} = \frac{f(x_{j+1}) - f(x_{j-1})}{2\Delta x}$$

$$f''(x_j) = \frac{\partial^2 f(x_j)}{\partial x^2} = \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1})}{\Delta x^2}$$

Discretizing our equation

Let us discretize the variable T in both time and space, with $T(t_i, x_j) = T_{i,j}$. This then gives us

$$\begin{aligned}
\frac{\partial T}{\partial t} &= k\frac{\partial^2 T}{\partial x^2}, \\
\frac{\partial T_{i,j}}{\partial t} &= k\frac{\partial^2 T_{i,j}}{\partial x^2}, \\
&= k\left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2}\right].
\end{aligned}$$

Discretizing our equation, continued

$$\frac{\partial T_{i,j}}{\partial t} = k\left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2}\right]$$

If we write $T_{i,j}$ as a vector of values, we can rewrite our equation as a matrix operation. Note that there is one equation for each spatial point:

$$\frac{\partial}{\partial t}\begin{bmatrix} \vdots \\ T_{i,17} \\ T_{i,18} \\ T_{i,19} \\ T_{i,20} \\ T_{i,21} \\ \vdots \end{bmatrix} = k\begin{bmatrix} & & & \vdots & & & \\ \cdots & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & 0 & \cdots \\ \cdots & 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \cdots \\ \cdots & 0 & 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & \cdots \\ & & & \vdots & & & \end{bmatrix}\begin{bmatrix} \vdots \\ T_{i,17} \\ T_{i,18} \\ T_{i,19} \\ T_{i,20} \\ T_{i,21} \\ \vdots \end{bmatrix}$$

Which we can write as $\frac{\partial \vec{T}_i}{\partial t} = F\vec{T}_i$.

What about the edges?

The edges are a problem. Why? Well, consider the first point, $j = 0$:

$$\frac{\partial T_{i,0}}{\partial t} = k \left[\frac{T_{i,1} - 2T_{i,0} + T_{i,-1}}{\Delta x^2} \right]$$

There is no $j = -1$ point!

The solution is to not use the above equation to describe the edge points. Use a different equation instead. These are known as 'boundary conditions'; there are two general classes:

- Dirichlet: constant boundary value.
- Neumann: boundary values based on derivatives of the boundary values themselves.

How these conditions are implemented depends on the approach to solving the equation that is being used.

Example problem

Suppose we have a rod, of length 1, whose temperature $T(t, x)$ is subject to the boundary conditions:

$$\begin{aligned} \sin(10t) \qquad & x = 0, \\ 0.0 \qquad & x = 1. \end{aligned}$$

The thermal diffusivity is $k = 0.2$. Assume there are $n = 100$ spatial points. Show how the temperature evolves in time and space.

How should we solve this problem? We could use explicit or implicit methods. Today we will just use explicit methods.

Using the explicit method

As mentioned previously, explicit methods can be unstable. Let's see how it performs in this case. This is the implicit version of the equation:

$$\frac{\vec{T}_{i+1} - \vec{T}_i}{\Delta t} = F\vec{T}_{i+1}.$$

This is the explicit version:

$$\begin{aligned} \frac{\vec{T}_{i+1} - \vec{T}_i}{\Delta t} &= F\vec{T}_i, \\ \vec{T}_{i+1} &= \vec{T}_i + \Delta t F\vec{T}_i, \\ \vec{T}_{i+1} &= (1 + \Delta t F)\,\vec{T}_i. \end{aligned}$$

Where $\mathbf{1}$ is the identity matrix.

Implementing boundary conditions

The boundary conditions are implemented by modifying the operator $\boldsymbol{F}$.

$$\vec{\boldsymbol{T}}_{i+1} = (1 + \boldsymbol{\Delta t F})\, \vec{\boldsymbol{T}}_i$$

The equation for the boundary condition at $\boldsymbol{j = 0}$ ($\boldsymbol{T_{i+1,0} = \sin(10 t_{i+1})}$) is implemented by:

$$\begin{bmatrix} T_{i+1,0} \\ T_{i+1,1} \\ T_{i+1,2} \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots \\ \alpha & 1-2\alpha & \alpha & 0 & 0 & \dots \\ 0 & \alpha & 1-2\alpha & \alpha & 0 & \dots \\ & & & \vdots & & \end{bmatrix} \begin{bmatrix} \sin(10 t_{i+1}) \\ T_{i,1} \\ T_{i,2} \\ \vdots \end{bmatrix}$$

Where we have used the definition: $\boldsymbol{\alpha = \Delta t k / (\Delta x^2)}$.

Implementing the other boundary condition

$$\vec{T}_{i+1} = (1 + \Delta t F)\, \vec{T}_i$$

We are assuming that there are 100 points in x. The boundary condition equation at $x = 1$ ($T_{i+1,99} = 0.0$) is implemented similarly:

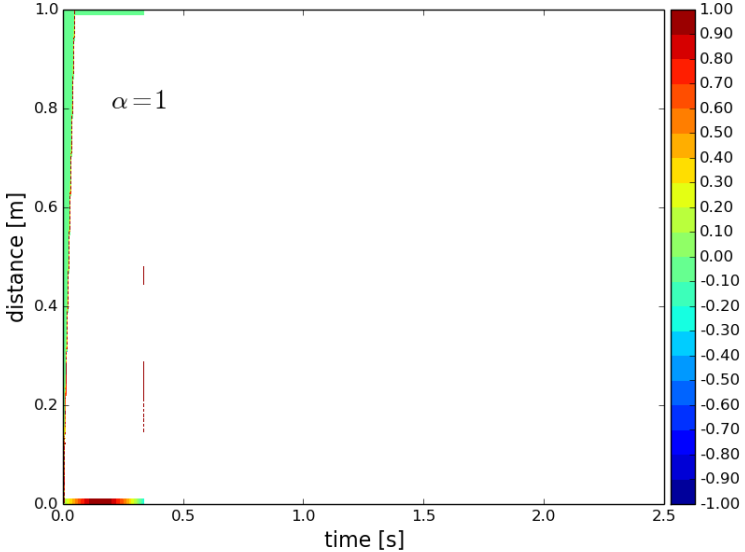
$$\begin{bmatrix} \vdots \\ T_{i+1,96} \\ T_{i+1,97} \\ T_{i+1,98} \\ T_{i+1,99} \end{bmatrix} = \begin{bmatrix} & & & \vdots & & \\ \dots & \alpha & 1-2\alpha & \alpha & 0 & 0 \\ \dots & 0 & \alpha & 1-2\alpha & \alpha & 0 \\ \dots & 0 & 0 & \alpha & 1-2\alpha & \alpha \\ \dots & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vdots \\ T_{i,97} \\ T_{i,97} \\ T_{i,98} \\ 0.0 \end{bmatrix}$$

Coding our example

```
#include <cmath>
#include <cblas.h>
int main() {
    double k = 0.2;
    double runtime = 2.0;
    int n = 100;
    double dx = 1.0 / (n - 1);
    double dt = 0.0005;
    int nsteps = runtime / dt;
    double a = 1.0, b = 0.0;
    double alpha = dt * k / dx**2;

    double *T = new double[n];
    double *rhs = new double[n];
    double **F = new double *[n];

    F[0] = new double[n * n];
    for(int i = 1; i < n; i++)
        F[i] = &F[0][i * n];
```

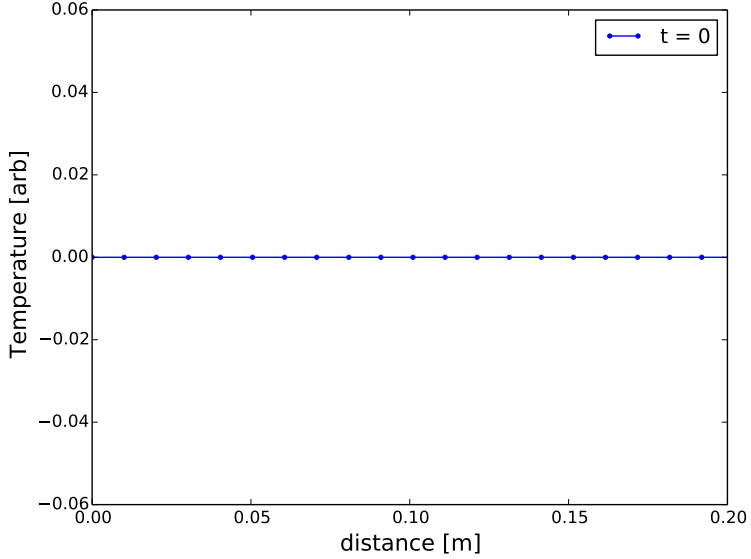
```
    for(int i = 0; i < n; i++) {
        T[i] = 0.0;     rhs[i] = 0.0;
        for(int j = 0; j < n; j++)
            F[i][j] = 0.0;}

    for (int i = 0; i < n; i++)
        if (i == 0) F[i][i] = 1.0;
        if ((i > 0) && (i < (n - 1))) {
            F[i][i-1] = alpha;  F[i][i+1] = alpha;
            F[i][i] = 1.0 - 2.0 * alpha;
        } F[i][i] = 1.0;

    for (int s = 1; i <= nsteps; s++) {
        double *temp = rhs;  rhs = T;  T = temp;
        rhs[0] = sin(dt * s * 10);
        rhs[n - 1] = 0.0;
        cblas_dgemv(CblasRowMajor, CblasNoTrans,
            n, n, a, F[0], n, rhs, 1, b, T, 1);
        // Output the result.
    }   // Deallocate.    return 0;}
```

Explicit results



There's a problem here.

What happened?



What went wrong?

Consider a 5-point version of the problem, given by the operator below.

$$\begin{bmatrix} T_{i+1,0} \\ T_{i+1,1} \\ T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & 0 \\ 0 & \alpha & 1-2\alpha & \alpha & 0 \\ 0 & 0 & \alpha & 1-2\alpha & \alpha \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sin(10t_{i+1}) \\ T_{i,1} \\ T_{i,2} \\ T_{i,3} \\ 0 \end{bmatrix}$$

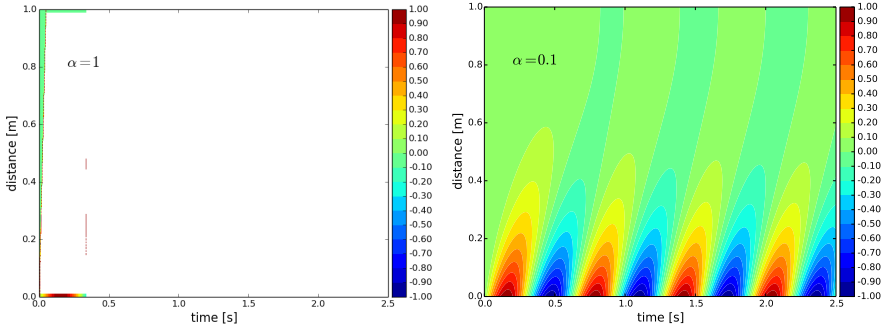
What are the conditions for this to be stable?

We require the eigenvalues of the operator $|\lambda| \leq 1$, and thus $\alpha = \frac{\Delta t k}{\Delta x^2} \leq 0.5$. Which gives

$$\Delta t \leq \frac{\Delta x^2}{2k}$$

If you double the spatial resolution, you need to quadruple the temporal resolution, for stability.

Can we save it?



Yes, we just need to pick a better value of α.

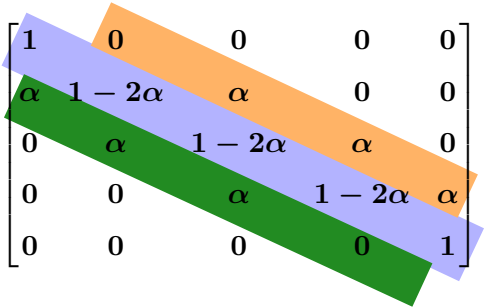
Notes about this implementation

A few things to recognize about this implementation:

- The code allocates and fills the entire matrix. Most of the matrix is zeros.
- This is a good way to implement things initially, for testing. It is not a good way to do things for production.
- Why? Because in this case the matrix is banded, and so a routine for a banded matrix should have been used to solve this problem.
- Who cares? Banded routines are faster and use much much less memory. Use them!
- Downside: banded matrices are a little more complicated to store, which is why we test them against the full matrix.

Banded-matrix storage

So how are banded matrices stored?



Finite volume methods

Finite volume methods are based on the discretization of the integral form of the equations, rather than the differential form.

- Convert all divergences into fluxes through the surfaces of each volume.
- Create a grid on which we will evaluate our fields. "Finite volume" refers to the small volume of space which surrounds each grid point. These are called "control volumes" (CVs).
- Discretize the equations on the CVs.
- Solve. Make sure your fluxes balance.

Finite volume methods, continued

So what have you done to the equations?

$$\frac{\partial T}{\partial t} = k\nabla^2 T = k\left(\nabla \cdot \nabla T\right)$$

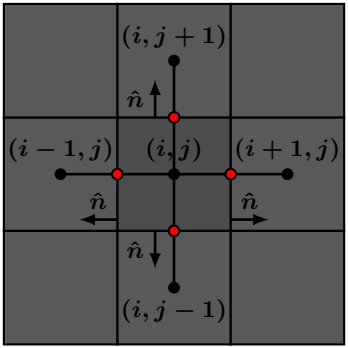
$$\begin{aligned} \frac{\partial}{\partial t}\int T dV &= k\int \left(\nabla \cdot \nabla T\right) dV \\ &= k\int \left(\nabla T \cdot \hat{n}\right) dS \end{aligned}$$

where we have used the divergence theorem, and $\hat{n}$ is the unit vector normal to the CV's surfaces.

Finite volume methods, continued

$$\frac{\partial}{\partial t}\int T dV = k\int (\nabla T\cdot\hat{n})\, dS$$

- T is calculated at the center, ∇T at the surfaces.
- Assume constant T in the CV and constant ∇T across the surface.
- Dot ∇T with the normal vectors $\hat{n}$.
- Here (i,j) refers to (x,y) indices.



$$\begin{aligned}\Delta x^2\frac{\partial T}{\partial t} &= k\left[\left(\frac{T_{i,j}-T_{i-1,j}}{\Delta x}\right)(-1)+\left(\frac{T_{i+1,j}-T_{i,j}}{\Delta x}\right)+\right.\\ &\quad\left.\left(\frac{T_{i,j}-T_{i,j-1}}{\Delta x}\right)(-1)+\left(\frac{T_{i,j+1}-T_{i,j}}{\Delta x}\right)\right]\Delta x\\ &= k\left(T_{i-1,j}+T_{i,j-1}+T_{i+1,j}+T_{i,j+1}-4T_{i,j}\right)\end{aligned}$$

Advantages of finite volume methods

Finite volume methods have a number of advantages:

- The methods ensure that all quantities are conserved, both locally and globally, assuming that all surface fluxes balance.
- The methods are easily adapted to irregular meshes, or unstructured meshes (arbitrary polyhedra or polygons).
- The methods can be especially powerful in cases where the mesh moves or adapts to the simulation.

For the case where the grid is uniform, the finite volume method reduces to the finite difference method, as seen on the previous slide.

Spectral methods

Spectral methods involve expanding fields in terms of some orthogonal basis set. Sinusoids are often use in situations which are periodic (periodic boxes):

$$T(t,x) = \sum_{n=0}^{n_{\max}} a_n(t) \sin\left(\frac{2\pi n x}{L}\right) + b_n(t) \cos\left(\frac{2\pi n x}{L}\right)$$

For $0 \leq x \leq L$.

If your geometry is spherical, you might expand in spherical harmonics:

$$T(t,r,\theta,\phi) = \sum_{l,m} a_{l,m}(t,r) Y_{l,m}(\theta,\phi)$$

You might also expand your r dependence spectrally:

$$T(t,r,\theta,\phi) = \sum_{l,m,\alpha} a_{l,m,\alpha}(t) N_\alpha(r) Y_{l,m}(\theta,\phi)$$

Why would you do that?

There are a number of reasons why you might solve your equations spectrally:

- The equations may be easier to solve, especially if the operators are eigenfunctions of the basis in question.
- The expansions are global in nature. If your boundary conditions are global (such as magnetic boundary condtions), this may be the only way to easily satisfy them.
- They can be very accurate, in certain cases converging exponentially to exact solutions.
- Because they are accurate, fewer grid points are needed, resulting in less memory being needed.
- If conversion between real-space and spectral-space is needed to be regularly performed as part of the algorithm, advanced algorithms exists for some of these transforms (FFTs).

Why wouldn't you do that?

Spectral methods have some downsides as well:

- The implementations can be more difficult to code.
- Because the expansions are global in nature, the domain needs to match the expansion basis. Complicated geometries lose accuracy, since the expansion must be long.
- Nonlinear terms can be very slow to be calculated.

SciNet Physics UNIVERSITY OF TORONTO