# Simulation data spectral/physical conversion, processing and storage

Martin Schreiber

05/10/16

## 1 Introduction

This document provides the information on the reasons how the simulation data is stored, processed and converted to/from spectral/physical space. The suggested changes are intended to get into SWEET within a few weeks after creating this document.

With the intended extension of supporting simulations on the plane with the double-Fourier and on the sphere with Spherical Harmonics, also the way how data is stored and processed should be made more clear. Several requirements arise from mathematics, 3rd party library and software design. After giving an overview on these requirements and discussion different ways of how to cope with the issue, we provide a conclusion in the end on the suggested changes in SWEET.

## 2 Requirements

### 2.1 Mathematics

1. Support for anti-aliasing
2. Support of different resolutions in spectral and physical space

### 2.2 Software framework

1. Support operator-styled arithmetic operations (e.g. a+b where a and b are data arrays)
2. Hide iteration loops and anti-aliasing if requested
3. Support sphere with SPH and plane data with FFT in a similar way

### 2.3 HPC

1. Efficient implementation on nowadays HPC shared-memory systems (e.g. caring about NUMA awareness)

2. No additional memory consumption

# 3 FFT vs. SPH

Converting data from spectral to physical space, we rely on existing libraries. Libraries for the FFTW have restrictions which is the reason for this discussion document.

## 3.1 Requirements for spectral/physical conversions

Let $(N, M)$ be the number of modes in spectral space and $(X, Y)$ the resolution in physical space. For the FFT used for simulations on the plane, we use the FFTW. For the SPH for simulations on the sphere we use the SHTns library. We directly encounter a restriction by using the FFTW. This only supports transformations if with $X = N$ and $Y = M$.

On the other hand, SHTns and also related libraries for spherical harmonics support different size of spectral and physical spaces with $X! = N$ and $Y! = M$.

## 3.2 Comment on "plans"

Both libraries require the creation of "plans". These plans should be created only once and then reused. A reference counter is used to free plans if they are not required anymore.

## 3.3 (Anti-)aliasing

### 3.3.1 No anti-aliasing

Without anti-aliasing it is sufficient to have the same number of modes in spectral space as the resolution is in physical space: $X = N$ and $Y = M$.

### 3.3.2 SPH Anti-aliasing

Anti-aliasing treatment is required for evaluating non-linearities. Even for a linear operators, such non-linearities might be required to multiply e.g. with $(1 - \mu^2)^{-1}$ which is done in physical space if using spherical harmonics. We only focus on implementations which support a fixed physical resolution for a given number of modes. (In the current implementation the physical space is always $X = N$ and $Y = M$ which might result in certain issues regarding anti-aliasing modes.)

Using SPH libraries, they typically support resolutions and a maximum number of modes which are not equivalent and we can easily support anti-aliasing.

Here, the 2/3 rule can be directly realized by using different sizes for the spatial/spectral space. Just by choosing a lower number of modes allows using only one buffer for spectral space of size $N \times M$ and another buffer for the data in spatial space of size $X \times Y$.

#### 3.3.3 FFTW anti-aliasing

However, FFTW requires to have identical spatial resolution and number of spectral modes, hence $X = N$ and $Y = M$. To cope with anti-aliasing, we require to run a FFT with a larger spectrum but with the higher modes set to zero before each of these FFTs. The size of the spectrum is given by $(NL, ML)$ and is typically $(NL, ML) = (N3/2, M3/2)$. We also discuss how to handle the operators (add, sub) in spectral space. We continue to discuss different methods to cope with this issue:

- **Two buffers with padding:**
  MUL: There are two separate buffers of size $(N, M)$ and $(NL, ML)$. Computing a multiplication in physical space is then accomplished by

  1. Copy data from $(N, M)$ to $(NL, ML)$ and use *zero padding.*
  2. Use FFT with $(NL = X, ML = Y)$ to compute representation in physical space
  3. [multiplication in physical space]
  4. Use inverse FFT with $(NL = X, ML = Y)$ to compute representation in spectral space
  5. Copy corresponding modes from $(NL, ML)$ to $(N, M)$.

  ADD/SUB: All other operators such as adding two spectral representations can be then realized by a single for loop without knowing about boundaries of $(N, M)$ and $(NL, ML)$ but just of $I = N \times M$.
  DRAWBACKS: Handling of two buffers and either always allocating an additional buffer or requiring to allocate a buffer if required.
  ADVANTAGES: Optimal application of spectral operators due to the single for loop.

- **Merged buffers to single one with padding:**
  This idea is basically the same one as the one before. However, the difference is that both buffers are shared.
  DRAWBACKS: Copy data from $(N, M)$ to $(NL, ML)$ and use zero padding might not be accomplished in parallel
  ADVANTAGES: Avoid using 2nd buffer, optimal application of spectral operators.

- **Use only single spectral buffer with special loop iterations:**
  Here, we allocate a buffer of size $(NL, ML)$ and handle all ADD/SUB of spectral data by only partial loop intervals $(N, M)$. Computing a multiplication in physical space is then accomplished by

1. Zero out modes ($> N, > M$).
2. Use FFT with ($NL = X, ML = Y$) to compute representation in physical space
3. [multiplication in physical space]
4. Use inverse FFT with ($NL = X, ML = Y$) to compute representation in spectral space

ADD/SUB: A special treatment is required to handle the iterations in spectral space in order to avoid iterating over ($NL, ML$) since only an iteration over ($N, M$) is required.
DRAWBACKS: Special iteration in spectral space required, maybe not optimal
ADVANTAGES: Only a single buffer, reassembles the SPH implementation

## 4 Suggestion (which gets the final decision)

We focus on changing the current implementation to the last suggested implementation to use only a single spectral buffer and a special loop seems to be the best decision.

- Mathematics: All requirements of the mathematical issues are met.
- Software framework: The software framework will get significantly simplified. This only requires support for a single plan for a particular resolution. All other requirements are also met.
- HPC: NUMA domains will be supported. There will be also no additional memory consumption for this.

The performance for the spectral operations might be less optimal but that's the price to pay to get clearer interfaces. Each x-related component (C storage layout) can be still handled in a single loop.

Additionally, once anti-aliasing is activated, the physical space follows entirely the 2/3 rule also for the other conversion (e.g. for input/output and setup routines) to the physical space.

This also makes **all** truncation functions exposed to the user obsolete!