How to Design an Efficient Pseudospectral Code

John C. Bowman
University of Alberta

June 11, 2019

www.math.ualberta.ca/~bowman/talks

Acknowledgements: Malcolm Roberts (Université de Strasbourg)
Outline

- Dealiased Pseudospectral Method
- 2D and 3D Skeleton ProtoDNS Codes on GitHub
- Key Ingredients for an Efficient Pseudospectral Solver:
  - Hermitian Symmetry (velocity and vorticity fields are real)
  - Implicit Dealiasing
  - Basdevant Reduction
  - Hybrid OpenMP/MPI Parallelization
  - Adaptive Time Stepping
  - Dynamic Moment Averaging
  - Conservative Integration
  - Implementation of White-Noise Forcing
- Conclusions
Pseudospectral Method

- Pseudospectral simulations are a widely used numerical tool for the study of fluid turbulence:
  - fast $N \log N$ scaling for $N$ modes.
  - spectral accuracy: more accurate than finite-difference or finite-element methods.

- Ideal choice for studying homogenous turbulence with periodic boundary conditions.

- Generalizations such as Chebyshev collocation and penalty methods allow them to handle more complicated boundary conditions and geometries.

- However, in many cases pseudospectral methods do not parallelize well on massively parallel distributed architectures due to the communication costs of the parallel transpose.
Dealiasing

• Patterson and Orszag pioneered the pseudospectral method over 40 years ago.

• They emphasized that the convolution theorem necessitates *dealiasing* unwanted harmonics arising from the periodicity of the discrete Fourier transform.
DNS code

- We have released a highly optimized 2D pseudospectral code in C++: [https://github.com/dealias/dns](https://github.com/dealias/dns).

- It uses our FFTW++ library to implicitly dealias the advective convolution, while exploiting Hermitian symmetry [Bowman & Roberts 2011], [Roberts & Bowman 2018].

- Advanced computer memory management, such as implicit padding, memory alignment, and dynamic moment averaging allow DNS to attain its extreme performance.

- The formulation proposed by Basdevant [1983] is used to reduce the number of FFTs required for 2D (3D) incompressible turbulence to 4 (8).

- We also include simplified 2D (146 lines) and 3D (287 lines) versions called ProtoDNS for educational purposes: [https://github.com/dealias/dns/tree/master/protodns](https://github.com/dealias/dns/tree/master/protodns).
Hermitian Symmetry

- In Fourier space, the reality of the velocity and scalar vorticity fields leads to the Hermitian symmetries

\[ \mathbf{v}_k = \overline{\mathbf{v}_k}, \]

\[ \omega_k = \overline{\omega_k}. \]

- The DC mode at the Fourier origin is not evolved (no mean flow).
Hermitian Symmetry in 2D

- Only the shaded modes need to be evolved:

\[
\begin{align*}
N_x &= 2m_x - 1 \\
N_y &= 2m_y - 1
\end{align*}
\]

- Warning: since Hermitian convolution routines require input from the entire blue rectangle, the white modes in the blue box must agree with their shaded Hermitian conjugates!
Discrete Cyclic Convolution

- The FFT is an efficient tool for computing the discrete cyclic convolution of two vectors $F$ and $G$ with period $N$:

$$\sum_{p=0}^{N-1} F_p G_{k-p}.$$

- But the pseudospectral method requires a linear convolution!

- The backward 1D discrete Fourier transform of a complex vector $\{F_k : k = 0, \ldots, N - 1\}$ is defined as

$$f_j \doteq \sum_{k=0}^{N-1} \zeta_N^{jk} F_k, \quad j = 0, \ldots, N - 1,$$

where $\zeta_N = e^{2\pi i/N}$ denotes the $N$th primitive root of unity.

- The fast Fourier transform (FFT) method exploits the properties that $\zeta_N^r = \zeta_{N/r}$ and $\zeta_N^N = 1$. 
Convolution Theorem

\[
\sum_{j=0}^{N-1} f_j g_j \zeta_N^{-jk} = \sum_{j=0}^{N-1} \zeta_N^{-jk} \left( \sum_{p=0}^{N-1} \zeta_N^{jp} F_p \right) \left( \sum_{q=0}^{N-1} \zeta_N^{jq} G_q \right)
\]

\[
= \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} F_p G_q \sum_{j=0}^{N-1} \zeta_N^{(-k+p+q)j}
\]

\[
= N \sum_{s} \sum_{p=0}^{N-1} F_p G_{k-p+sN}.
\]

- The terms indexed by \(s \neq 0\) are aliases; we need to remove them!

- If only the first \(m\) entries of the input vectors are nonzero, aliases can be avoided by zero padding input data vectors of length \(m\) to length \(N \geq 2m - 1\).
Explicit Dealiasing

- *Explicit zero padding* prevents mode $m - 1$ from beating with itself, wrapping around to contaminate mode $N = 0 \mod N$:
Explicit Dealiasing

- *Explicit zero padding* prevents mode $m - 1$ from beating with itself, wrapping around to contaminate mode $N = 0 \mod N$:

\[
\begin{align*}
\{F_k\}_{k=0}^{N-1} & \quad \{0\}_{k=0}^{N-1} \\
\{G_k\}_{k=0}^{N-1} & \quad \{0\}_{k=0}^{N-1}
\end{align*}
\]
Explicit Dealiasing

- *Explicit zero padding* prevents mode \( m - 1 \) from beating with itself, wrapping around to contaminate mode \( N = 0 \mod N \):

\[
\begin{align*}
\{F_k\}_{k=0}^{N-1} & \quad \{0\}_{k=0}^{N-1} \\
\{f_j\}_{j=0}^{2N-1} & \\
\{G_k\}_{k=0}^{N-1} & \quad \{0\}_{k=0}^{N-1} \\
\{g_j\}_{j=0}^{2N-1} &
\end{align*}
\]
Explicit Dealiasing

- **Explicit zero padding** prevents mode $m - 1$ from beating with itself, wrapping around to contaminate mode $N = 0 \mod N$:

\[
\begin{align*}
\{F_k\}_{k=0}^{N-1} & \rightarrow \{0\}_{k=0}^{N-1} \\
\{f_j\}_{j=0}^{2N-1} & \rightarrow \{f_jg_j\}_{j=0}^{2N-1} \\
\{G_k\}_{k=0}^{N-1} & \rightarrow \{0\}_{k=0}^{N-1} \\
\{g_j\}_{j=0}^{2N-1} & \rightarrow \{0\}_{j=0}^{2N-1}
\end{align*}
\]
Implicit Dealiasing

• Let $N = 2m$. For $j = 0, \ldots, 2m - 1$ we want to compute

$$f_j = \sum_{k=0}^{2m-1} \zeta_{2m}^j F_k.$$

• If $F_k = 0$ for $k \geq m$, one can easily avoid looping over the unwanted zero Fourier modes by decimating in wavenumber:

$$f_{2\ell} = \sum_{k=0}^{m-1} \zeta_{2m}^{2\ell k} F_k = \sum_{k=0}^{m-1} \zeta_{m}^{\ell k} F_k, \quad \ell = 0, 1, \ldots, m - 1.$$

$$f_{2\ell+1} = \sum_{k=0}^{m-1} \zeta_{2m}^{(2\ell+1) k} F_k = \sum_{k=0}^{m-1} \zeta_{m}^{\ell k} \zeta_{2m}^k F_k.$$ 

• This requires computing two subtransforms, each of size $m$, for an overall computational scaling of order $2m \log_2 m = N \log_2 m$. 
Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer FFTW++ (v2.05) on top of the FFTW library under the Lesser GNU Public License:

http://fftwpp.sourceforge.net/

\[
\{F_k\}_{k=0}^{m-1} \quad \{G_k\}_{k=0}^{m-1}
\]
Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer FFTW++ (v2.05) on top of the FFTW library under the Lesser GNU Public License:

http://fftwpp.sourceforge.net/

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{G_k\}_{k=0}^{m-1} \\
\{f_{2\ell}\}_{\ell=0}^{m-1} & \quad \{f_{2\ell+1}\}_{\ell=0}^{m-1} \\
\{g_{2\ell}\}_{\ell=0}^{m-1} & \quad \{g_{2\ell+1}\}_{\ell=0}^{m-1}
\end{align*}
\]
Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer FFTW++ (v2.05) on top of the FFTW library under the Lesser GNU Public License:

http://fftwpp.sourceforge.net/

**Diagram:**

- \( \{F_k\}_{k=0}^{m-1} \)
- \( \{G_k\}_{k=0}^{m-1} \)
- \( \{f_{2\ell}\}_{\ell=0}^{m-1} \)
- \( \{f_{2\ell+1}\}_{\ell=0}^{m-1} \)
- \( \{g_{2\ell}\}_{\ell=0}^{m-1} \)
- \( \{g_{2\ell+1}\}_{\ell=0}^{m-1} \)
- \( \{f_{2\ell}g_{2\ell}\}_{\ell=0}^{m-1} \)
- \( \{f_{2\ell+1}g_{2\ell+1}\}_{\ell=0}^{m-1} \)
Parallelized multidimensional implicit dealiasing routines have been implemented as a software layer FFTW++ (v2.05) on top of the FFTW library under the Lesser GNU Public License:

http://fftwpp.sourceforge.net/
**Input:** vector $f$, vector $g$

**Output:** vector $f$

$$u \leftarrow \text{fft}^{-1}(f);$$
$$v \leftarrow \text{fft}^{-1}(g);$$
$$u \leftarrow u \ast v;$$

for $k = 0$ to $m - 1$ do
  $$f[k] \leftarrow \zeta^k_{2m} f[k];$$
  $$g[k] \leftarrow \zeta^k_{2m} g[k];$$
end

$$v \leftarrow \text{fft}^{-1}(f);$$
$$f \leftarrow \text{fft}^{-1}(g);$$
$$v \leftarrow v \ast f;$$

$$f \leftarrow \text{fft}(u);$$
$$u \leftarrow \text{fft}(v);$$

for $k = 0$ to $m - 1$ do
  $$f[k] \leftarrow f[k] + \zeta^{-k}_{2m} u[k];$$
end

return $f/(2m)$;
Implicit Padding in 1D on T threads

![Graph showing time/(m log2 m) (ns) vs. m with different thread counts (T=1 and T=4) for explicit and implicit padding.](image-url)
Convolutions in Higher Dimensions

• An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

• An explicitly padded convolution in 2 dimensions requires 12 padded FFTs and 4 times the memory of a cyclic convolution.
Recursive Convolution

• Naive way to compute a multiple-dimensional convolution:

\[ F_{N_1, \ldots, N_d} \rightarrow \text{multiply} \rightarrow F_{N_1, \ldots, N_d}^{-1} \]

• The technique of recursive convolution allows one to avoid computing and storing the entire Fourier image of the data:

\[ F_{N_d} \rightarrow N_d \times \text{convolve}_{N_1, \ldots, N_{d-1}} \rightarrow F_{N_d}^{-1} \]
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.
Implicit Padding in 2D on T threads

\[ \text{time}/(m^2 \log_2 m^2) \text{ (ns)} \]

- △ explicit T=1
- □ implicit T=1
- ⦿ explicit T=4
- ⦿ implicit T=4
Implicit Padding in 3D on T threads

![Graph showing time/(m^3 log_2 m^3) (ns) vs m, with different line styles for explicit and implicit padding with T=1 and T=4.]
Centered (Pseudospectral) Convolutions

- For a centered convolution, the Fourier origin ($k = 0$) is centered in the domain:

$$
\sum_{p=k-m+1}^{m-1} f_p g_{k-p}
$$

- Need to pad to $N \geq 3m - 2$ to remove aliases.

- The ratio $(2m - 1)/(3m - 2)$ of the number of physical to total modes is asymptotic to $2/3$ for large $m$.

- A Hermitian convolution arises since the input vectors are real:

$$
f_{-k} = \overline{f_k}.
$$
Hermitian Convolution

- The backwards implicitly padded centered Hermitian transform appears as

\[ u_{3\ell+r} = \sum_{k=0}^{m-1} \zeta_{m}^{\ell k} w_{k,r}, \]

where

\[ w_{k,r} = \begin{cases} 
U_0 + \text{Re} \zeta_3^{-r} U_{-m} & \text{if } k = 0, \\
\zeta_3^{r k} (U_k + \zeta_3^{-r} U_{m-k}) & \text{if } 1 \leq k \leq m - 1.
\end{cases} \]

- We exploit the Hermitian symmetry \( w_{k,r} = \overline{w_{m-k,r}} \) to reduce the problem to three complex-to-real Fourier transforms of the first \( c+1 \) components of \( w_{k,r} \) (one for each \( r = -1, 0, 1 \)), where \( c \overset{\text{def}}{=} \lfloor m/2 \rfloor \) zeros.
To facilitate an in-place implementation, in our original paper [SIAM J. Sci. Comput. 33, 386 (2011)], we stored the transformed values for $r = 1$ in reverse order in the upper half of the input vector.

However, loop dependencies in the resulting algorithm prevented the top level of the 1D transforms from being multithreaded.
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.

\[ r = 0 \]
\[
\begin{array}{c}
0 \\
1 \\
\cdots \\
k \\
\cdots \\
c + 1 - k \\
\cdots \\
c - 1 \\
c \\
\cdots \\
c + k \\
\cdots \\
2c + 1 - k \\
\cdots \\
2c - 1 \\
2c \\
\end{array}
\]
As a result, even in 1D, implicit dealiasing of pseudospectral convolutions is now significantly faster than explicit zero padding.
1D Implicit Hermitian Convolution

\[
\text{time}/(m \log_2 m) \text{ (ns)}
\]

\[
\begin{array}{c}
10^2 & 10^3 & 10^4 & 10^5 & 10^6 \\
\text{m} & \text{explicit T=1} & \text{implicit T=1} & \text{explicit T=4} & \text{implicit T=4}
\end{array}
\]
3D Basdevant Reduction: 8 FFTs

- Using incompressibility, the 3D momentum equation can be written in terms of the symmetric tensor $D_{ij} = u_i u_j$:

\[
\frac{\partial u_i}{\partial t} + \frac{\partial D_{ij}}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} + F_i.
\]

- Naive implementation: 3 backward FFTs to compute the velocity components from their spectral representations, 6 forward FFTs of the independent components of $D_{ij}$.

- Basdevant [1983]: avoid one FFT by subtracting the divergence of the symmetric matrix $S_{ij} = \delta_{ij} \text{tr} D/3$ from both sides:

\[
\frac{\partial u_i}{\partial t} + \frac{\partial(D_{ij} - S_{ij})}{\partial x_j} = -\frac{\partial(p\delta_{ij} + S_{ij})}{\partial x_j} + \nu \frac{\partial^2 u_i}{\partial x_j^2} + F_i.
\]

- To compute the velocity components $u_i$, 3 backward FFTs are required. Since the symmetric matrix $D_{ij} - S_{ij}$ is traceless, it has just 5 independent components.
• Hence, a total of only 8 FFTs are required per integration stage.

• The effective pressure $p\delta_{ij} + S_{ij}$ is solved as usual from the inverse Laplacian of the force minus the nonlinearity.
2D Basdevant Reduction: 4 FFTs

• The vorticity $w = \nabla \times u$ evolves according to

$$\frac{\partial w}{\partial t} + (u \cdot \nabla)w = (w \cdot \nabla)u + \nu \nabla^2 w + \nabla \times F,$$

where in 2D the vortex stretching term $(w \cdot \nabla)u$ vanishes and $w$ is normal to the plane of motion.

• For $C^2$ velocity fields, the curl of the nonlinearity can be written in terms of $\tilde{D}_{ij} = D_{ij} - S_{ij}$:

$$\frac{\partial}{\partial x_1} \frac{\partial}{\partial x_j} \tilde{D}_{2j} - \frac{\partial}{\partial x_2} \frac{\partial}{\partial x_j} \tilde{D}_{1j} = \left( \frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2} \right) D_{12} + \frac{\partial}{\partial x_1} \frac{\partial}{\partial x_2} (D_{22} - D_{11}),$$

on recalling that $S$ is diagonal and $S_{11} = S_{22}$.

• The scalar vorticity $\omega$ thus evolves as

$$\frac{\partial \omega}{\partial t} + \left( \frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2} \right) (u_1 u_2) + \frac{\partial^2}{\partial x_1 \partial x_2} (u_2^2 - u_1^2) = \nu \nabla^2 \omega + \frac{\partial F_2}{\partial x_1} - \frac{\partial F_1}{\partial x_2}.$$
• To compute $u_1$ and $u_2$ in physical space, we need 2 backward FFTs.

• The quantities $u_1 u_2$ and $u_2^2 - u_1^2$ can then be calculated and then transformed to Fourier space with 2 additional forward FFTs.

• The advective term in 2D can thus be calculated with just 4 FFTs.
3D Incompressible MHD: 14 FFTs

\[
\frac{\partial u_i}{\partial t} + \frac{\partial (D_{ij} - S_{ij})}{\partial x_j} = -\frac{\partial (p\delta_{ij} + S_{ij})}{\partial x_j} + \nu \frac{\partial^2 u_i}{\partial x_j^2},
\]

\[
\frac{\partial B_i}{\partial t} + \frac{\partial G_{ij}}{\partial x_j} = \eta \frac{\partial^2 B_i}{\partial x_j^2},
\]

where \(D_{ij} = u_iu_j - B_iB_j\), \(S_{ij} = \delta_{ij} \text{tr } D/3\), and

\[G_{ij} = B_iu_j - u_iB_j.\]

- The traceless symmetric matrix \(D_{ij} - S_{ij}\) has 5 independent components.
- The antisymmetric matrix \(G_{ij}\) has only 3.
- An additional 6 FFT calls are required to compute the components of \(\mathbf{u}\) and \(\mathbf{B}\) in \(x\) space.
- The MHD nonlinearity can thus be computed with 14 FFT calls.
2D Navier–Stokes Pseudospectral [1 thread]

\[
\frac{\text{time}}{(m \log_2 m)^2} \text{ (s)}
\]

- Explicit
- Implicit

\[
10^{2} \quad 10^{3}
\]

\[
m
\]
2D Navier–Stokes Pseudospectral [4 threads]
Distributed-Memory Parallelization

- The pseudospectral method uses a matrix transpose to localize the computation of the multi-dimensional FFTs onto individual processors.

- Parallel generalized slab/pencil decompositions have recently been developed for distributed-memory architectures.

- We have compared several distributed matrix transpose algorithms, both blocking and nonblocking, under pure MPI and hybrid MPI/OpenMP architectures.

- Local transposition is not required within a single MPI node.

- We have developed an adaptive algorithm, dynamically tuned to choose the optimal block size.
8 × 8 Block Transpose over 8 processors

<table>
<thead>
<tr>
<th>Process</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$8 \times 8$ Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
$8 \times 8$ Block Transpose over 8 processors
$8 \times 8$ Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
Advantages of Hybrid MPI/OpenMP

- Use hybrid OpenMPI/MPI with the optimal number of threads:
  - yields larger communication block size;
  - local transposition is not required within a single MPI node;
  - allows smaller problems to be distributed over a large number of processors;
    - for 3D FFTs, allows for more slab-like than pencil-like models, reducing the size of or even eliminating the need for a second transpose.
  - sometimes more efficient (by a factor of 2) than pure MPI.

- The use of nonblocking MPI communications allows us to overlap computation with communication: this can yield up to an additional 32% performance gain for implicitly dealiased convolutions, for which a natural parallelism exists between communication and computation.
2D Forced–Dissipative Turbulence Spectrum

\[ E(k) \]

- \( k_H = 300 \)
- \( k_H = 0 \)
2D Forced–Dissipative Power Law Exponent

The logarithmic slope of $E(k)$ is shown as a function of $k$ for two different values of $k_H$: $k_H = 300$ (red line) and $k_H = 0$ (blue line). The graph highlights the power-law behavior with exponents $-5$, $-4$, $-3$, and $-2$.
Conservative Integration

- Conservative integration [SIAM J. Appl. Math 59, 1112 (1999)] provides a useful diagnostic technique for ensuring that the underlying dynamical symmetries have been correctly implemented.
White-Noise Forcing

- The Fourier transform of an isotropic Gaussian white-noise solenoidal force \( \mathbf{f} \) has the form

\[
\mathbf{f}_k(t) = F_k \left( 1 - \frac{k k}{k^2} \right) \cdot \mathbf{\xi}_k(t), \quad \mathbf{k} \cdot \mathbf{f}_k = 0,
\]

where \( F_k \) is a real number and \( \mathbf{\xi}_k(t) \) is a unit central real Gaussian random 2D vector that satisfies

\[
\langle \mathbf{\xi}_k(t) \mathbf{\xi}_k'(t') \rangle = \delta_{kk'} 1 \delta(t - t').
\]

- This implies

\[
\langle \mathbf{f}_k(t) \cdot \mathbf{f}_k'(t') \rangle = F_k^2 \delta_{k,k'} \delta(t - t').
\]
White-Noise Forcing

- The rate of energy injection $\epsilon$ is given by

$$\epsilon = (f(x, t), u(x, t)) = \int_\Omega \langle f(x, t) \cdot u(x, t) \rangle \, dx = \text{Re} \sum_k \langle f_k(t) \cdot \bar{u}_k(t) \rangle$$

- Here $u_k(t)$ is functional of the forcing:

$$u_k(t) = u_k(t') + \int_{t'}^{t} A_k[u(\tau)] \, d\tau + \int_{t'}^{t} f_k(\tau) \, d\tau,$$

where $A_k$ is a functional of $u$ such that $\frac{\delta A_k[u(\tau)]}{\delta f_k(t')}$ is bounded.

- Nonlinear Green’s function:

$$\frac{\delta u_k(t)}{\delta f_{k'}(t')} = \int_t^{t'} \frac{\delta A_k[u(\tau)]}{\delta f_{k'}(t')} \, d\tau + \delta_{k k'} 1 H(t - t'),$$

where $H$ is the Heaviside unit step function.
To prescribe the forcing amplitude $F_k$ in terms of $\epsilon$:

**Theorem 1** (Novikov [1964]): If $f(\mathbf{x}, t)$ is a Gaussian process, and $u$ is a functional of $f$, then

$$\langle f(\mathbf{x}, t)u(f) \rangle = \int \int \langle f(\mathbf{x}, t)f(\mathbf{x}', t') \rangle \left\langle \frac{\delta u(\mathbf{x}, t)}{\delta f(\mathbf{x}', t')} \right\rangle d\mathbf{x}' dt'.$$

For white-noise forcing:

$$\epsilon = \text{Re} \sum_k \langle f_k(t) \cdot \overline{u}_k(t) \rangle = \text{Re} \sum_{k,k'} \int \langle f_k(t) \overline{f}_{k'}(t') \rangle : \left\langle \frac{\delta \overline{u}_k(t)}{\delta \overline{f}_{k'}(t')} \right\rangle dt'$$

$$= \sum_k F_k^2 \left( 1 - \frac{kk}{k^2} \right) : \left( 1 - \frac{kk}{k^2} \right) H(0)$$

$$= \frac{1}{2} \sum_k F_k^2,$$

on noting that $H(0) = 1/2$. 
Implementation of White-Noise Forcing

- At the end of each time-step, we implement the contribution of white noise forcing with the discretization

\[ \omega_{k,n+1} = \omega_{k,n} + \sqrt{2\tau \eta_k} \xi, \]

where \( \xi \) is a unit complex Gaussian random number with \( \langle \xi \rangle = 0 \) and \( \langle |\xi|^2 \rangle = 1 \).

- This yields the mean enstrophy injection

\[ \frac{\langle |\omega_{k,n+1}|^2 - |\omega_{k,n}|^2 \rangle}{2\tau} = \eta_k. \]
Conclusions

- For centered convolutions in $d$ dimensions, implicit padding asymptotically uses $(2/3)^{d-1}$ of the conventional storage.
- The factor of 2 speedup is largely due to increased data locality.
- Highly optimized and parallelized implicit dealiasing routines have been implemented as a software layer FFTW++ (v 2.05) on top of the FFTW library and released under the Lesser GNU Public License: http://fftwpp.sourceforge.net/
- The advent of implicit dealiasing of convolutions makes overlapping transposition with FFT computation feasible.
- Dynamic moment averaging allows the integration time window to be specified by the user *a posteriori*.
- Writing a high-performance dealiased pseudospectral code is now a relatively straightforward exercise: skeleton 2D and 3D optimized codes are available at https://github.com/dealias/dns/tree/master/protodns.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Details</th>
</tr>
</thead>
</table>