Scientific Computation

Spring, 2019

Lecture 16

Imperial College
London

Notes

- HW3 will be posted Thursday, ~7pm, due 21/3
- If you have strong feelings (positive or negative) about the midnight deadline, please let me know (or use the anonymous online feedback form)
- Feedback for HW2 will be provided before 21/3

Today:

- Wrap up discussion of DFTs
- Numerical differentiation (with finite differences)

This week's lab: Use DFTs for numerical differentiation

DFT notes

- We describe the amplitudes of a Fourier coefficient as the “energy” at a particular frequency
- Parseval’s formula provides some intuition for this description:

$$\sum_{j=0}^{N-1} |f_j|^2 = N \sum_{n=-N/2}^{N/2-1} |c_n|^2$$

- $|c_n|^2$ is the ***energy spectral density***
- For periodic data, computing the energy spectrum is straightforward
- For aperiodic data, some care is needed

Discrete Fourier transform

Now, our function is represented on a N-point discrete, equispaced grid:

$$t_j = j\Delta t,\ j = 0, 1, ..., N-1$$

We have a truncated Fourier series at the jth point:

$$f(t_j) = \sum_{n=-N/2}^{N/2-1} c_n exp(i2\pi n t_j/T) = \sum_{n=-N/2}^{N/2-1} c_n exp(i2\pi jn/N)$$

with $N\Delta t = T$**, and the inverse transform is now a discrete sum:**

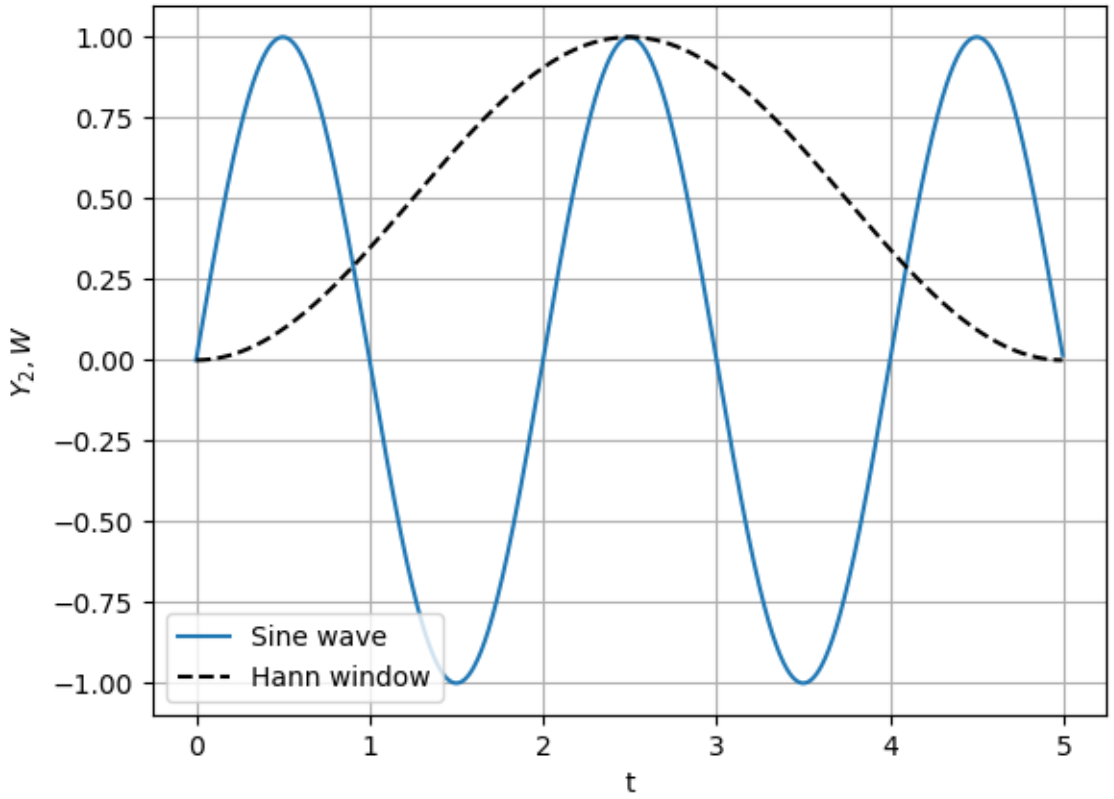
$$c_n = \frac{1}{N}\sum_{j=0}^{N-1} f_j exp(-i2\pi n t_j/T) = \frac{1}{N}\sum_{j=0}^{N-1} f_j exp(-i2\pi jn/N)$$

- np.fft.fft **computes** c**, with the (1/N) factor omitted, but returns it in a … strange order:**

$$\mathrm{np.fft.fft(f)} = \mathrm{c_{np}} = \mathrm{N}\left(\mathrm{c_0}, \mathrm{c_1}, ..., \mathrm{c_{N/2-1}}, \mathrm{c_{-N/2}}, \mathrm{c_{-N/2+1}}, ..., \mathrm{c_{-1}}\right)$$

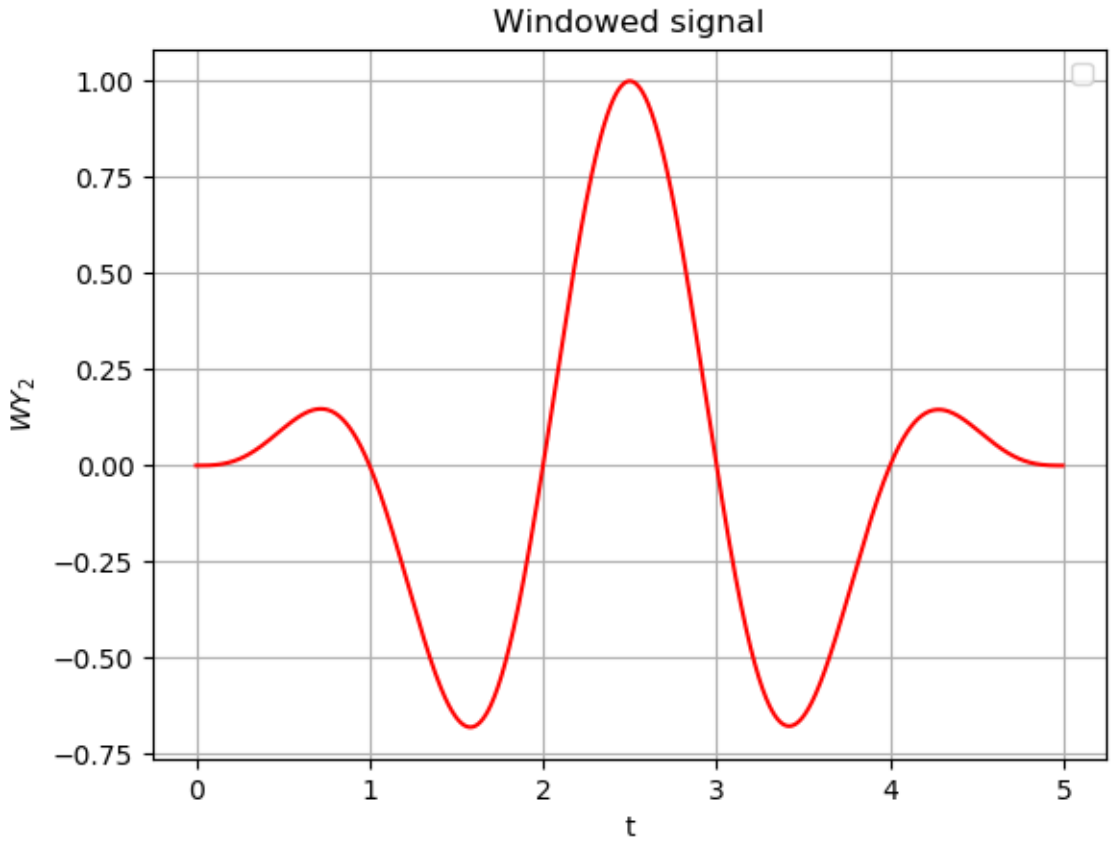
Aperiodic data

- There is no reason to generally expect signals to be periodic so this is an important issue
- The standard “fix” is to use windowing:
-



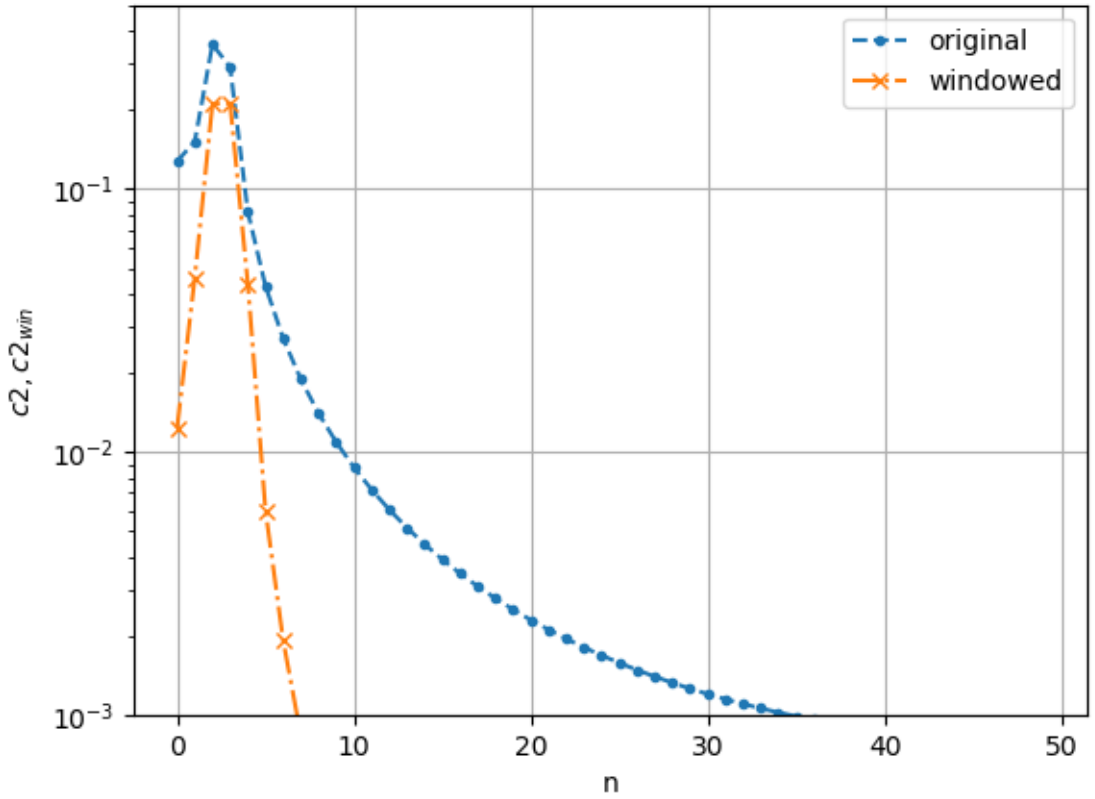
Aperiodic data

- There is no reason to generally expect signals to be periodic so this is an important issue
- The standard “fix” is to use windowing:
-



Aperiodic data

- There is no reason to generally expect signals to be periodic so this is an important issue
- The standard “fix” is to use windowing:
- The spectrum of the windowed signal “looks” more like a simple wave
- We have lost energy – there is no perfect solution
-



Data analysis

- In practice, we don't have to go through the windowing process ourselves
- Signal processing tools exist which:
  - Break the signal up into overlapping segments
  - Window the signal within each segment and compute the spectrum
  - Average the spectra from each segment (typically $|c|^2$ rather than $|c|$)
- This produces an estimate of the *power spectral density*
- And can be computed using *Welch's method*, scipy.signal.welch

```
In [201]: w2,Pxx2 = sig.welch(Y2)

In [203]: w2 = w2*Nt/T

In [206]: plt.semilogy(w2,Pxx2)
```

Data analysis

- In practice, we don't have to go through the windowing process ourselves
- Signal processing tools exist which:
  - Break the signal up into overlapping segments
  - Window the signal within each segment and compute the spectrum
  - Average the spectra from each segment (typically $|c|^2$ rather than $|c|$)
- This produces an estimate of the *power spectral density*
- And can be computed using *Welch's method*, scipy.signal.welch

```
In [201]: w2,Pxx2 = sig.welch(Y2)

In [203]: w2 = w2*Nt/T

In [206]: plt.semilogy(w2,Pxx2)
```

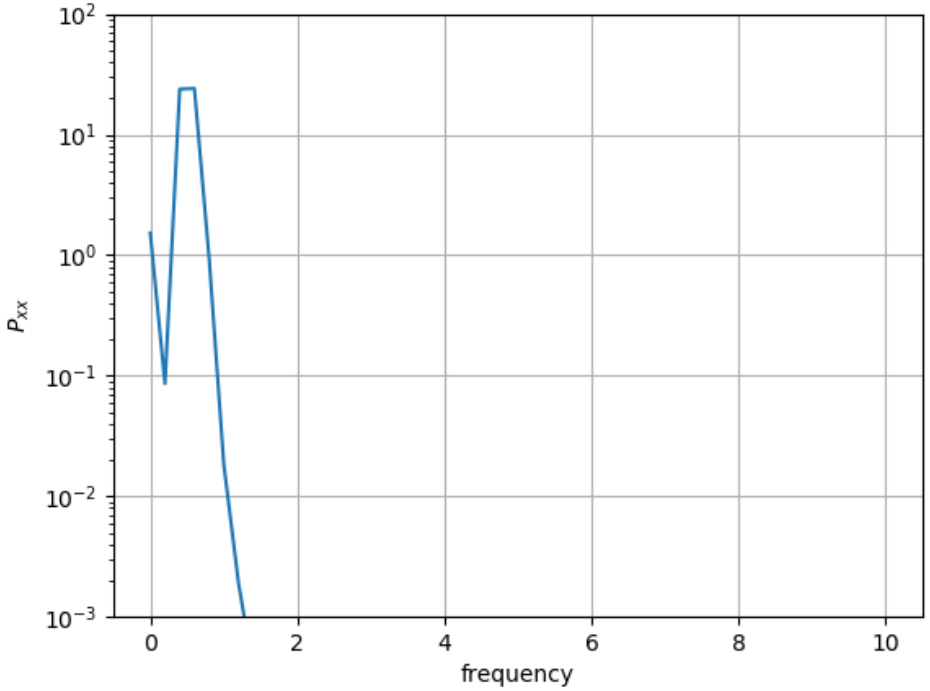
Data analysis

And can be computed using *Welch's method*, scipy.signal.welch

```
In [201]: w2,Pxx2 = sig.welch(Y2)

In [203]: w2 = w2*Nt/T

In [206]: plt.semilogy(w2,Pxx2)
```



Data analysis

Notes:

- **Consider a signal of length T with step Δt as a distribution of "energy" across a range of frequencies**
- **We need $\Delta t < 2/f_{min}$ to *resolve* the highest frequency components**
- **Also need $T >> 1/f_{max}$ to ensure "slow" components are contained within the signal**
  - **Not the case in our previous example**
  - **Often, slow components have the largest amplitude**
- **This discussion applies to *stationary* processes:**
  - **All joint probabilities of finding the system at some time in one state and at some later time in another state are independent of time within the observation period (they only depend on the time *difference*)**
  - **Weak stationarity: Mean and variance are independent of time**

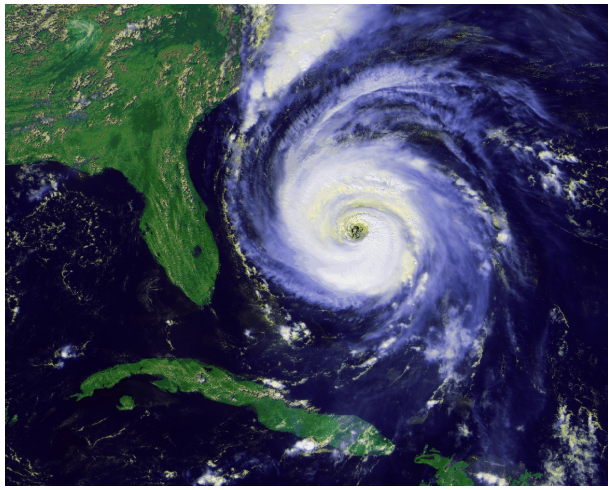
Data analysis

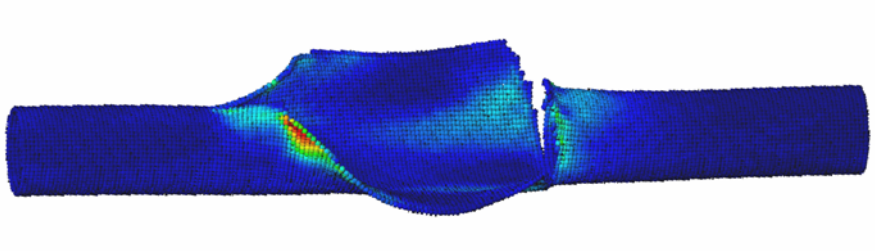
Notes:

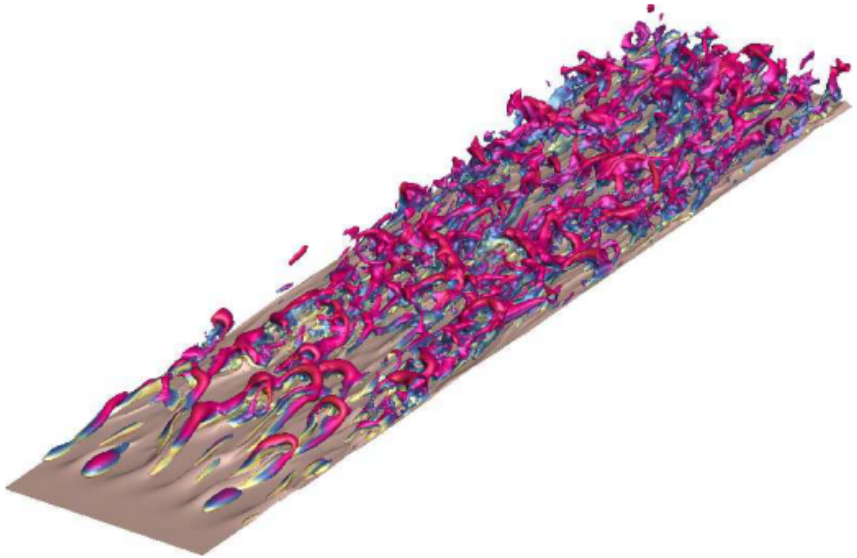
- **Consider a signal of length T with step Δt as a distribution of "energy" across a range of frequencies**
- **We need $\Delta t < 2/f_{min}$ to *resolve* the highest frequency components**
- **Also need $T >> 1/f_{max}$ to ensure "slow" components are contained within the signal**
  - **Not the case in our previous example**
  - **Often, slow components have the largest amplitudes**
- **What about the running time?**
  - **Direct evaluation of the sum in the DFT requires $O(N^2)$ operations**
  - **But the FFT uses a divide and conquer approach and $O(N\log_2 N)$ operations are needed**

Multiscale science and engineering

- **Climate modelling, aerodynamics, material simulation → All highly nonlinear, multiscale, very complex**

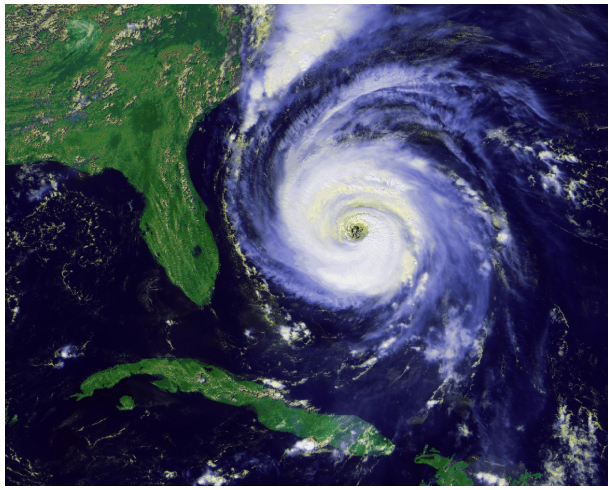


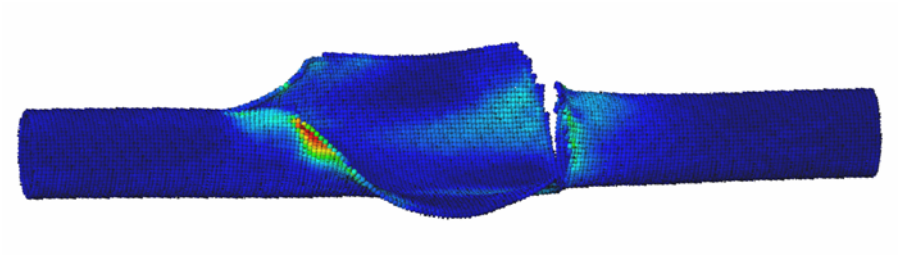


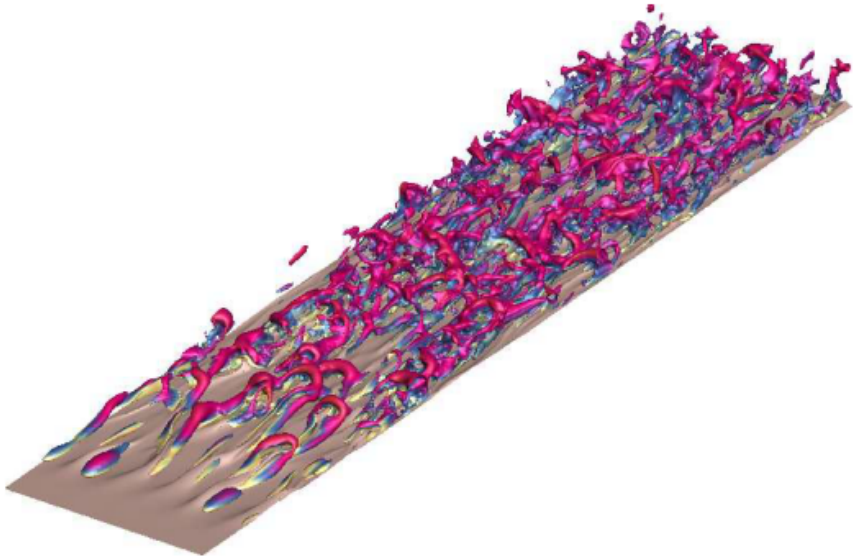


Multiscale science and engineering

- **Climate modelling, aerodynamics, material simulation → All highly nonlinear, multiscale, very complex**
- **Which numerical methods are best?**

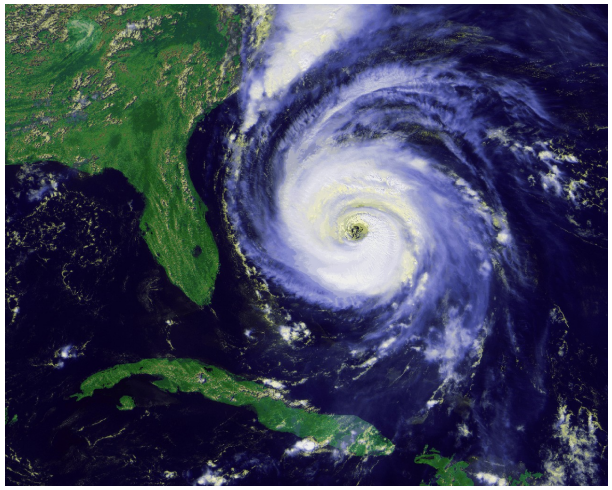


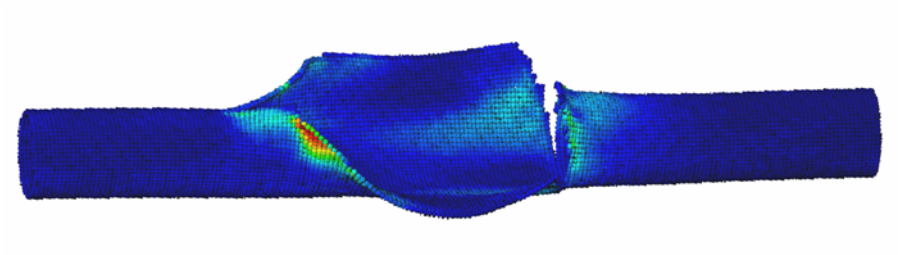


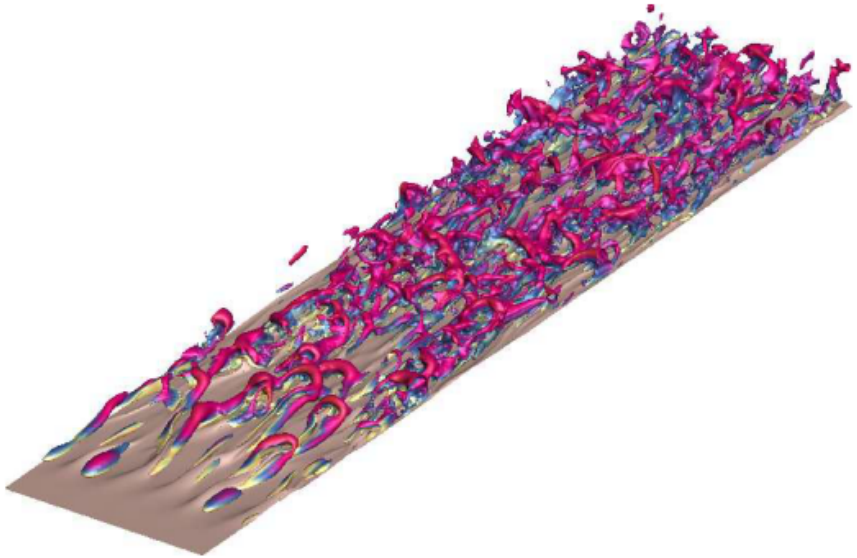


Multiscale science and engineering

- Climate modelling, aerodynamics, material simulation → All highly nonlinear, multiscale, very complex
- Which numerical methods are best?
- Today: Analyzing and designing finite difference methods for multiscale problems







2nd-order finite difference

- Centered approximation for first derivative
- Uniform grid with grid spacing, ***h***

$$f_i' = \frac{(f_{i+1} - f_{i-1})}{2h} + \text{truncation error}$$

2nd-order finite difference

- Centered approximation for first derivative
- Uniform grid with grid spacing, ***h***

$$f_i' = \frac{(f_{i+1} - f_{i-1})}{2h} + \text{truncation error}$$

- Truncation error? Taylor series →

$$f_{i+1} = f_i + hf_i' + \frac{h^2}{2} f_i'' + \frac{h^3}{6} f_i''' + \ldots$$

$$f_{i-1} = f_i - hf_i' + \frac{h^2}{2} f_i'' - \frac{h^3}{6} f_i''' + \ldots$$

Review: 2nd-order finite difference

- Truncation error? Taylor series →

$$f_{i+1} = f_i + hf_i' + \frac{h^2}{2} f_i'' + \frac{h^3}{6} f_i''' + \ldots$$

$$f_{i-1} = f_i - hf_i' + \frac{h^2}{2} f_i'' - \frac{h^3}{6} f_i''' + \ldots$$

- Subtract 2nd equation from first,

$$f_i' = \frac{(f_{i+1} - f_{i-1})}{2h} + \frac{h^2}{6} f_i''' + \ldots$$

- Usually say, "error is $O(h^2)$"
- But there is more information in the truncation error…

Wavenumber analysis

- Consider 'wave,' $f(x) = e^{ikx}$
- and its derivative, $f' = ike^{ikx}$

Wavenumber analysis

- Consider 'wave,' $f(x) = e^{ikx}$
- and its derivative, $f' = ike^{ikx}$

- Finite difference approximation of f':

$$f_{i+1} = e^{ik(x+h)}$$

$$f_{i-1} = e^{ik(x-h)}$$

$$\frac{(f_{i+1} - f_{i-1})}{2h} = \frac{e^{ikh} - e^{-ikh}}{2h} e^{ikx}$$

$$\frac{(f_{i+1} - f_{i-1})}{2h} = i \left[\frac{\sin(kh)}{h} \right] e^{ikx}$$

Wavenumber analysis

- Consider 'wave,' $f(x) = e^{ikx}$
- and its derivative, $f' = ike^{ikx}$
- Finite difference approximation of f':

$$f_{i+1} = e^{ik(x+h)}$$

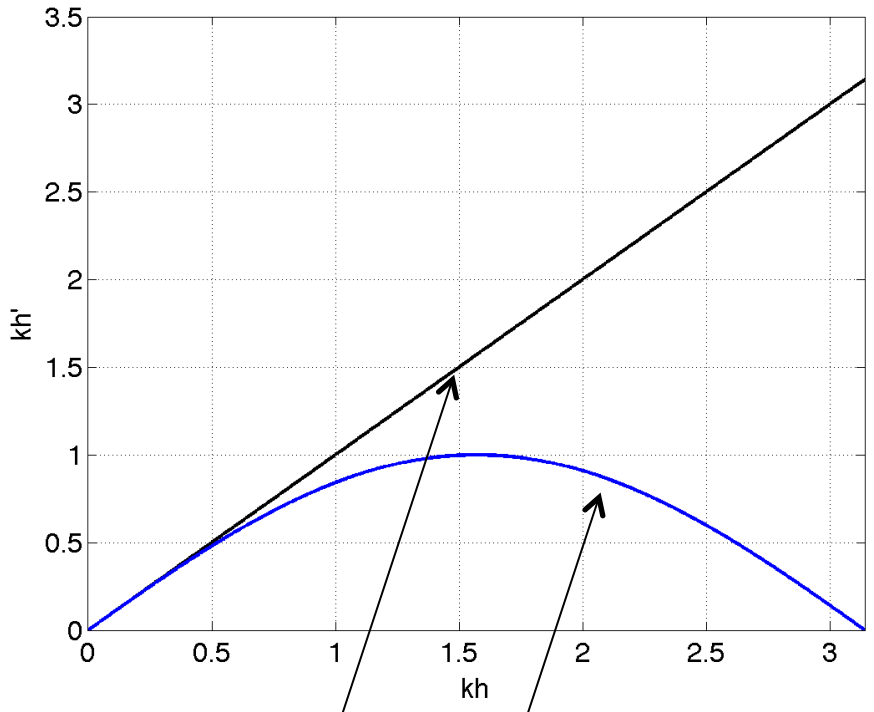
$$f_{i-1} = e^{ik(x-h)}$$

$$\frac{(f_{i+1} - f_{i-1})}{2h} = \frac{e^{ikh} - e^{-ikh}}{2h} e^{ikx}$$

$$\frac{(f_{i+1} - f_{i-1})}{2h} = i \left[\frac{\sin(kh)}{h} \right] e^{ikx}$$

Wavenumber analysis

- **Compare 'modified wavenumber,' $\sin(kh)$ to kh**



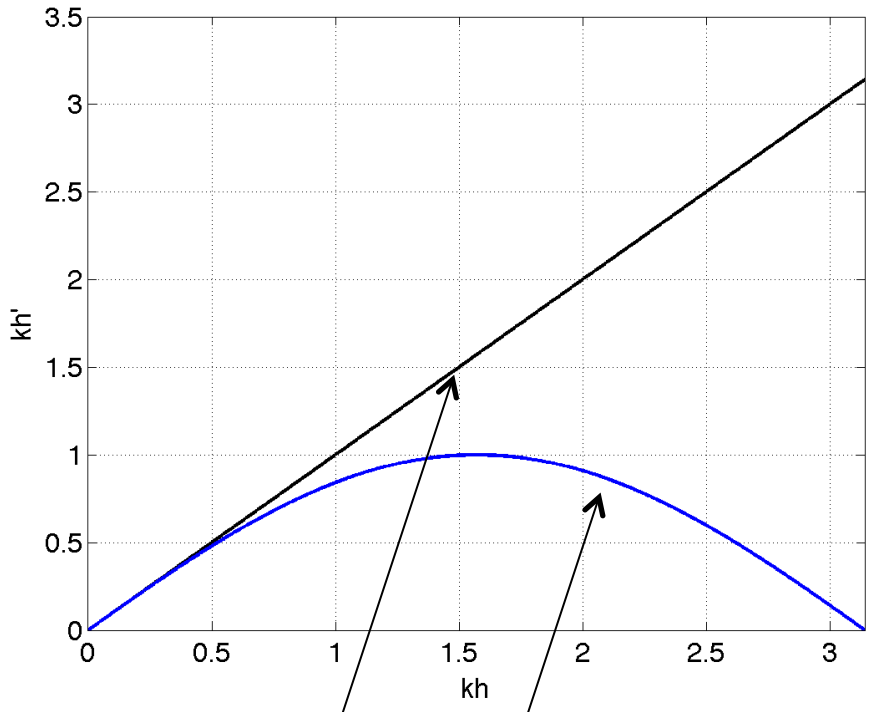
- **FD only accurate for low wavenumbers (long waves)**

Black: exact

Blue: 2nd-order finite difference

Wavenumber analysis

- **Compare 'modified wavenumber,' $\sin(kh)$ to kh**



Black: exact

Blue: 2nd-order finite difference

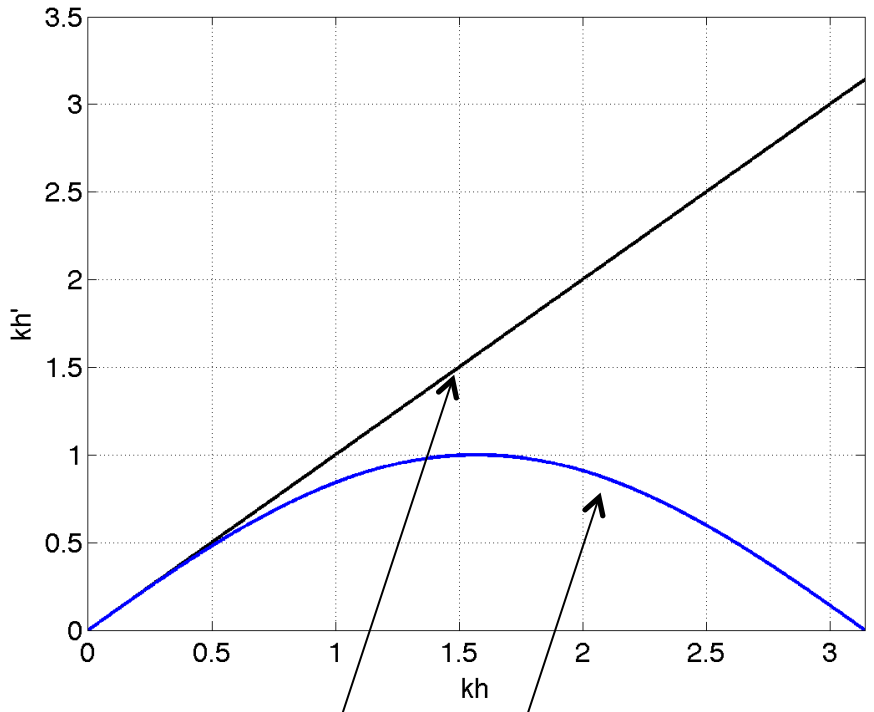
- **FD only accurate for low wavenumbers (long waves)**
- **1% error when** $kh \approx 0.39$

$$kh = \frac{2\pi h}{\lambda}$$

$$\lambda/h \approx 16$$

Wavenumber analysis

- **Compare 'modified wavenumber,' $\sin(kh)$ to kh**



Black: exact

Blue: 2nd-order finite difference

- **FD only accurate for low wavenumbers (long waves)**
- **1% error when** $kh \approx 0.39$

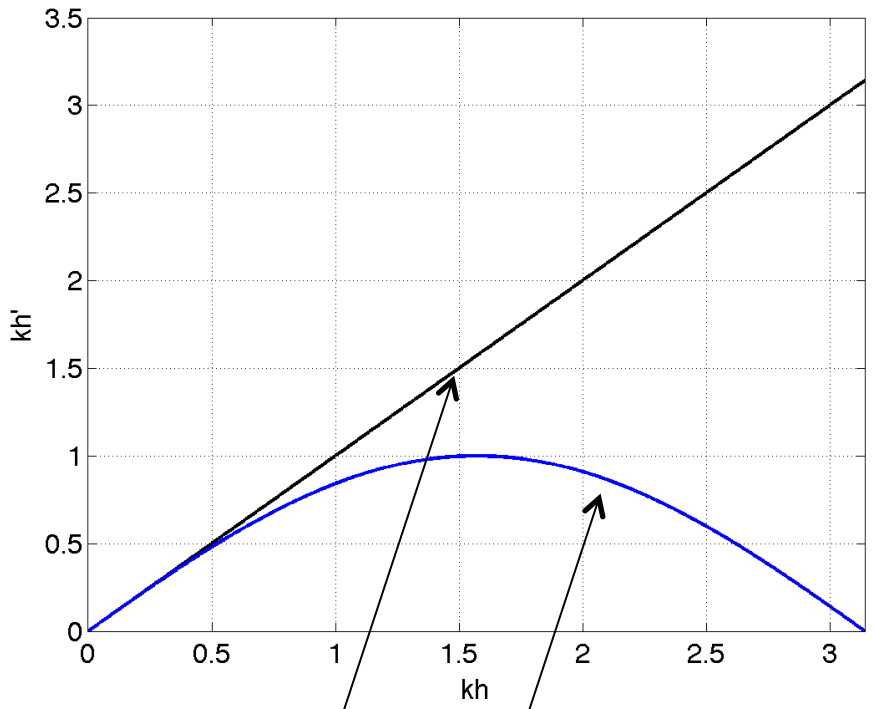
$$kh = \frac{2\pi h}{\lambda}$$

$$\lambda/h \approx 16$$

- **So, need ~16 points/wavelength for ~1 % error**
- **Spectral (Fourier): >2 points/ wavelength, exact**

Wavenumber analysis

- **Compare 'modified wavenumber,' $\sin(kh)$ to kh**



Black: exact

Blue: 2nd-order finite difference

- **FD only accurate for low wavenumbers (long waves)**
- **1% error when** $kh \approx 0.39$

$$kh = \frac{2\pi h}{\lambda}$$

$$\lambda/h \approx 16$$

- **So, need ~16 points/wavelength for ~1 % error**
- **Spectral (Fourier): >2 points/ wavelength, exact**

Exercise: What is modified wavenumber for 2nd derivative?:

$$f_i'' = \frac{(f_{i+1} - 2f_i + f_{i-1})}{h^2}$$

FD schemes with better resolution?

- **Think about time marching:**
  - **Explicit Euler has poor stability properties**
  - **Implicit Euler: much better stability, but requires matrix inversion → more expensive**

- **Explicit finite difference:** $f'_i = \frac{(f_{i+1} - f_{i-1})}{2h}$

- **Or 'implicit' finite difference stencils:**

$$\beta f'_{i-2} + \alpha f'_{i-1} + f'_i + \alpha f'_{i+1} + \beta f'_{i+2} =$$

$$c\frac{f_{i+3} - f_{i-3}}{6h} + b\frac{f_{i+2} - f_{i-2}}{4h} + a\frac{f_{i+1} - f_{i-1}}{2h}$$

- **Now have a system of equations (a *banded* matrix)**
- **Lots of coefficients to play with!**

$$\begin{bmatrix} f_1 & g_1 & h_1 & & & & \\ e_2 & f_2 & g_2 & h_2 & & & \\ d_3 & e_3 & f_3 & g_3 & h_3 & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & d_{n-1} & e_{n-1} & f_{n-1} & g_{n-1} \\ & & & & & d_n & e_n & f_n \end{bmatrix}$$

Designing your FD stencil

$$\beta f'_{i-2} + \alpha f'_{i-1} + f'_i + \alpha f'_{i+1} + \beta f'_{i+2} =$$

$$c\frac{f_{i+3} - f_{i-3}}{6h} + b\frac{f_{i+2} - f_{i-2}}{4h} + a\frac{f_{i+1} - f_{i-1}}{2h}$$

- **First think about truncation error:**
  - **Taylor series + lots of algebra →**

2nd order: $a + b + c = 1 + 2\alpha + 2\beta$

4th order: $a + 2^2 b + 3^2 = \frac{4!}{2!}(\alpha + 2^2\beta)$

6th order: $a + 2^4 b + 3^4 c = \frac{6!}{4!}(\alpha + 2^4\beta)$

Designing your FD stencil

2nd order: $a + b + c = 1 + 2\alpha + 2\beta$

4th order: $a + 2^2 b + 3^2 = \frac{4!}{2!}(\alpha + 2^2\beta)$

6th order: $a + 2^4 b + 3^4 c = \frac{6!}{4!}(\alpha + 2^4\beta)$

- **Highest possible accuracy: 10th order:**

$$\alpha = \frac{1}{2}, \beta = \frac{1}{20}, a = \frac{17}{12}, b = \frac{101}{150}, c = \frac{1}{100}$$

- **But is this the best scheme?**

Wavenumber analysis

$$\beta f'_{i-2} + \alpha f'_{i-1} + f'_i + \alpha f'_{i+1} + \beta f'_{i+2} =$$

$$c\frac{f_{i+3}-f_{i-3}}{6h} + b\frac{f_{i+2}-f_{i-2}}{4h} + a\frac{f_{i+1}-f_{i-1}}{2h}$$

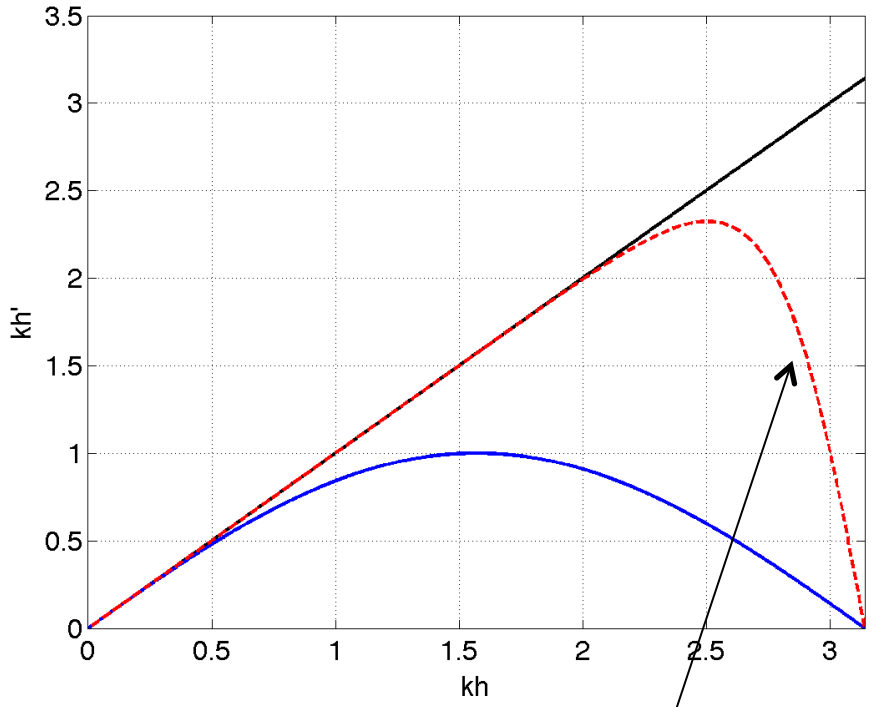
- **Same approach as before,** $f(x) = e^{ikx}$

$$f_{i+1} = e^{ik(x+h)}$$

gives the general modified wavenumber:

$$kh' = \frac{a\sin(kh)+(b/2)\sin(2kh)+(c/3)\sin(3kh)}{1+2\alpha\cos(kh)+2\beta\cos(2kh)}$$

Wavenumber analysis



- Previous result for 2nd order FD
- 10th order ~ 3 points/wavelength for 1% error
- But can we do better?

Black: exact

Blue: 2nd-order finite difference

Red: 10th-order finite difference

Wavenumber analysis

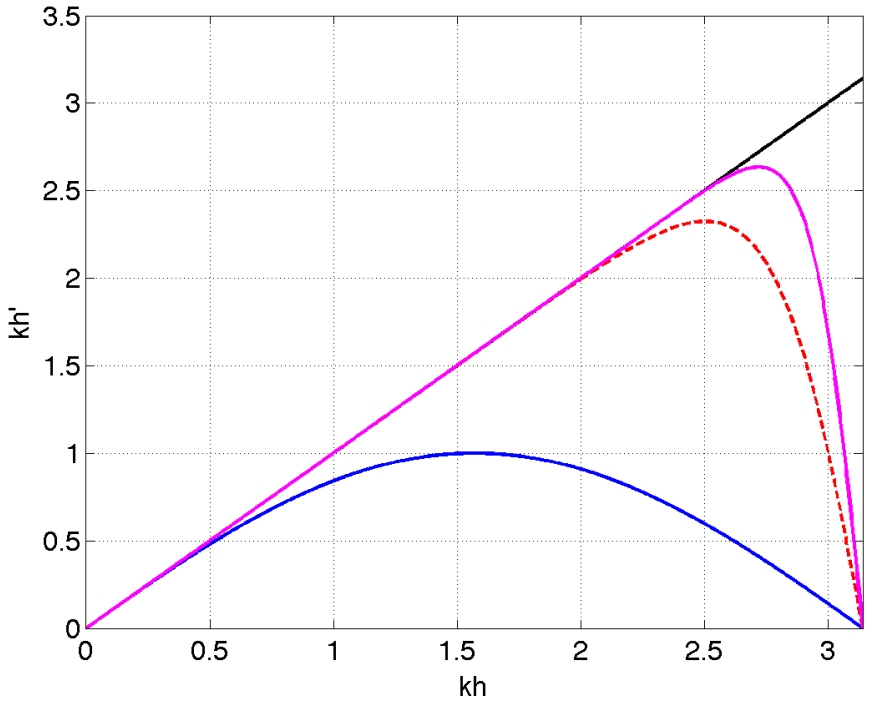
- 4th order schemes have three free constants
- Can impose constraints on modified wavenumber:
  - *e.g.:* $kh'(2.2) = 2.2$

    $kh'(2.3) = 2.3$

    $kh'(2.4) = 2.4$
- Then,

  $\alpha = 0.5771439, \beta = 0.0896406, a = 1.3025166, b = 0.99335500, c = 0.03750245$

Wavenumber analysis



- **10th order ~ 3 points/wavelength for 1% error**
- **4th order ~ 2.5 points/wavelength for 1% error**
- **Order of accuracy isn't everything!**

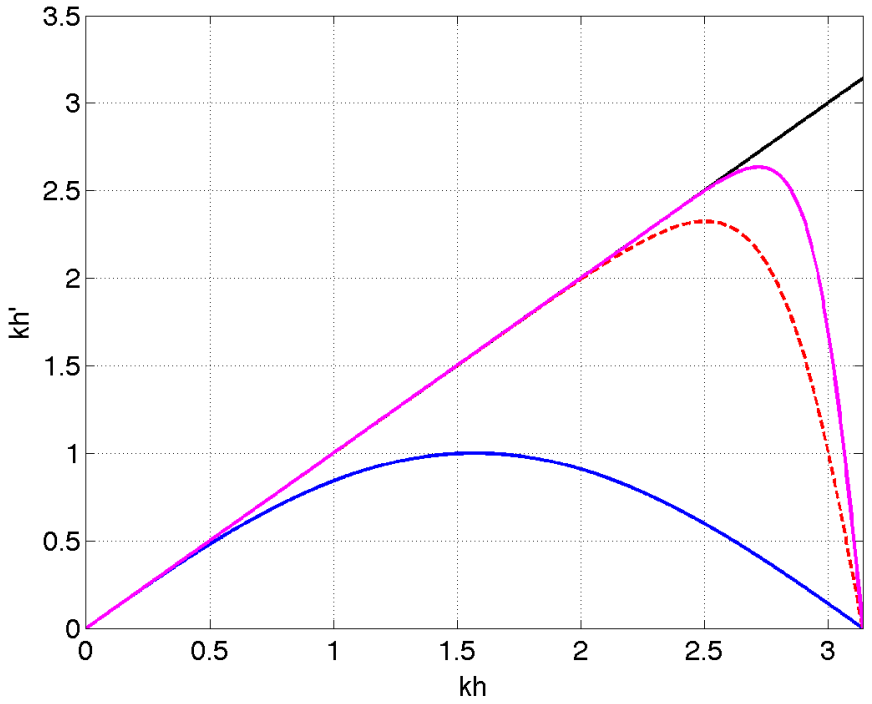
Black: exact

Blue: 2nd-order

Red: 10th-order

Magenta: Optimised, 4th order

Wavenumber analysis



Black: exact

Blue: 2nd-order

Red: 10th-order

Magenta: Optimised, 4th order

- 10th order ~ 3 points/wavelength for 1% error
- 4th order ~ 2.5 points/wavelength for 1% error
- Order of accuracy isn't everything!

- Notes on cost: operation counts→
  - 2nd order FD: N mult + N add
  - 4th order implicit FD: pentadiagonal linear system,
  - 7N mult + 7N add
  - Spectral (Fourier): ~N$\log_2$N

Key question: which method requires least time for desired accuracy?

Python implementation

- **Spectral-like scheme requires solution of** $\mathbf{Ax} = \mathbf{b}$ **and A is a pentadiagonal matrix:**

$$\begin{bmatrix} f_1 & g_1 & h_1 & & & & & \\ e_2 & f_2 & g_2 & h_2 & & & & \\ d_3 & e_3 & f_3 & g_3 & h_3 & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & d_{n-1} & e_{n-1} & f_{n-1} & g_{n-1} \\ & & & & & d_n & e_n & f_n \end{bmatrix}$$

- **In general,** np.solve(A,b)

- **But this isn't very efficient**
  - **Requires excessive memory (storing the zeros in** A**)**
  - **Requires excessive operations (again due to the zeros)**

Python implementation

- **Can build sparse banded matrix using** scipy.diags
- **First construct diagonals:**

```
#RHS
a = 1.3025166
b = 0.9935500
c = 0.03750245

#LHS
ag = 0.5771439
bg = 0.0896406


#Construct A
zv = np.ones(N)
agv = ag*zv[2:]
bgv = bg*zv[1:]
```

Python implementation

- **Then construct sparse matrix,** A:

```
A = sp.diags([agv,bgv,zv,bgv,agv],[-2,-1,0,1,2])
A.toarray()
```

```
Out[100]:
array([[1.       , 0.0896406, 0.5771439, 0.       , 0.       , 0.       ],
       [0.0896406, 1.       , 0.0896406, 0.5771439, 0.       , 0.       ],
       [0.5771439, 0.0896406, 1.       , 0.0896406, 0.5771439, 0.       ],
       [0.       , 0.5771439, 0.0896406, 1.       , 0.0896406, 0.5771439],
       [0.       , 0.       , 0.5771439, 0.0896406, 1.       , 0.0896406],
       [0.       , 0.       , 0.       , 0.5771439, 0.0896406, 1.       ]])
```

and use scipy.sparse.linalg.spsolve

- **This would solve the memory issue**
- **But it is still possible to do better with efficiency**

Python implementation

- scipy.linalg **has a solver specifically for banded matrices:** scipy.linalg.solve_banded **(actually a Fortran routine from Lapack)**

- **This is a little tricky to use**
  - **Need to provide matrix in "matrix diagonal ordered form"**

    $A_b[u + i - j, j] == A[i,j]$

  - Ab **is a 1-D array required as input to** solve_banded

  - u**, is the number of non-zero diagonals *above* the main diagonal (2 for our pentadiagonal matrix)**

- **Now we (nearly) have the "best" approach for our problem**
  - **Need to modify our method for the top two and bottom two rows in the matrix (i=0,1,N-2,N-1)**

Boundary modifications

At i=1 and i=N-2, we switch to a (8th-order) tridiagonal scheme:

$$\alpha = \tfrac{3}{8}. \qquad \beta = 0, \quad a = \tfrac{1}{6}(\alpha + 9),$$
$$b = \tfrac{1}{15}(32\alpha - 9), \quad c = \tfrac{1}{10}(-3\alpha + 1).$$

And at i=0,N-1, we switch to "one-sided" schemes:

$$f_0 + \alpha f_1 = \tfrac{1}{h}(a f_0 + b f_1 + c f_2 + d f_3)$$
$$f_{N-1} + \alpha f_{N-2} = -\tfrac{1}{h}(a f_{N-1} + b f_{N-2} + c f_{N-3} + d f_{N-4})$$

$$\alpha = 3, \quad a = -\tfrac{17}{6}, \quad b = \tfrac{3}{2},$$
$$c = \tfrac{3}{2}, \quad d = -\tfrac{1}{6}. \quad \text{(fourth order)}$$

This works for *periodic* functions where the RHS can be evaluated using these schemes. For general functions, we would have to modify the scheme at i=3,N-2 as well

Final notes

- **Advantages of quasi-spectral FD schemes**
  - **‘Competitive’ efficiency**
  - **Low memory usage (relative to explicit FD and Fourier) – second key question: how much memory is available? Is there a performance gain from using less memory? (low space complexity)**
- **Disadvantages**
  - **Global (like spectral)**
  - **Difficult to apply to complex geometries**