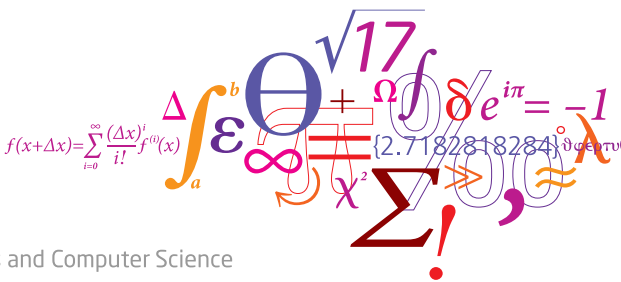
# Lecture 1 Fourier Methods

02689 Advanced Numerical Methods for Differential Equations

Allan Peter Engsig-Karup
Scientific Computing Section
DTU Compute
Technical University of Denmark




DTU Compute
Department of Applied Mathematics and Computer Science

# Fourier methods

General Fourier Series representation of a function $f(x)$ on a domain $x \in [0, T]$ where solution is periodic, i.e. $f(x) = f(x + nT)$ with $n$ an integer, for a real-valued function

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx)$$

with coefficients defined from

$$a_n = \frac{2}{T} \int_0^T f(x) \cos(nx) dx$$
$$b_n = \frac{2}{T} \int_0^T f(x) \sin(nx) dx$$

If $f(x)$ is symmetric, i.e. $f(-x) = f(x)$ then it is possible to show that $b_n = 0$ and if $f(x)$ is antisymmetric then $a_n = 0$.

## Fourier methods

The Fourier series can be expressed in complex form as

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

where the Euler formula has been used

$$e^{inx} = \cos(nx) + i\sin(nx), \quad i = \sqrt{-1}$$

and where the expansion coefficient are then defined from

$$c_n = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-inx}dx = \begin{cases} \frac{1}{2}a_0 & , n = 0 \\ \frac{1}{2}(a_n - ib_n) & , n > 0 \\ \frac{1}{2}(a_n + ib_n) & , n < 0 \end{cases}$$

With the given definitions we can find the inverse relationships

$$\begin{aligned} a_n &= c_n + c_{-n}, \quad n \geq 0 \\ b_n &= i(c_n - c_{-n}), \quad n \geq 1 \end{aligned}$$

## Trigonometric polynomials

For practical purposes the Fourier series is usually approximated by a truncated version (i.e. partial sum) of the form

$$\mathcal{P}_N f(x) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nx) + \sum_{n=1}^{N} b_n \sin(nx)$$

which is referred to as a real trigonometric polynomial of degree $N$.

This partial sum can be rewritten using Euler's formula and expressed as the projection

$$\mathcal{P}_N f(x) = c_0 + \sum_{n=1}^{N} \left( c_n e^{inx} + c_{-n} e^{-inx} \right) = \sum_{n=-N}^{N} c_n e^{inx}$$

If we introduce $\tau$ as the truncation error, then we find

$$f(x) = \mathcal{P}_N f(x) + \tau, \quad \tau = \sum_{|n|=N+1}^{\infty} c_n e^{inx}$$

with the error dependent on how rapid $c_n$ decays for $n \to \infty$.

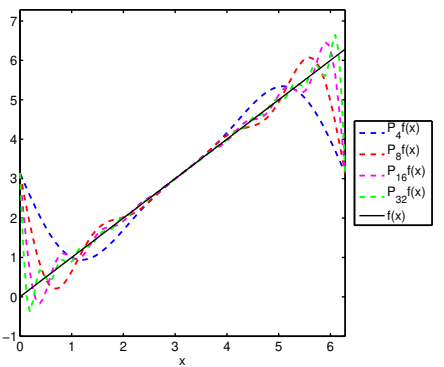
# Convergence of Fourier series

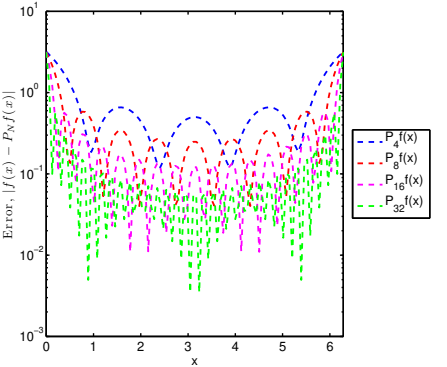
Consider the approximation of the periodic extension of the piece-wise continuous function

$$f_1(x) = x$$

which can be shown to have the expansion coefficients

$$c_n = \begin{cases} \pi & , n = 0 \\ \frac{i}{n} & , n \neq 0 \end{cases}$$



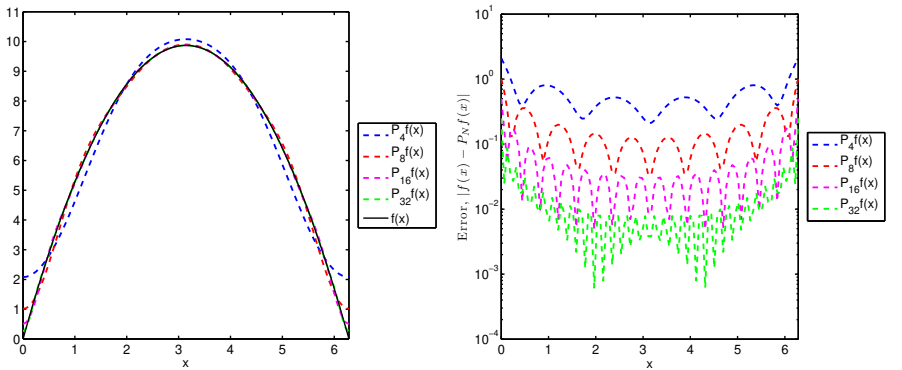
## Convergence of Fourier series

Consider the approximation of the $C_p^0([0, 2\pi])$ function

$$f_2(x) = x(2\pi - x)$$

which can be shown to have the expansion coefficients

$$c_n = \begin{cases} \frac{2\pi^2}{3} & , n = 0 \\ -\frac{2}{n^2} & , n \neq 0 \end{cases}$$



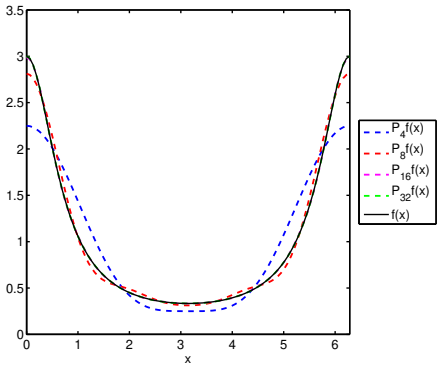
## Convergence of Fourier series

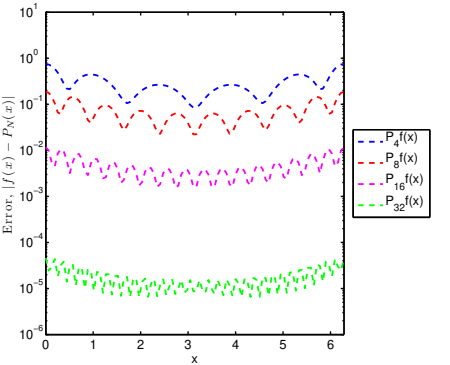
Consider the approximation of the $C_p^\infty([0, 2\pi])$ function

$$f_3(x) = \frac{3}{5 - 4\cos(x)}$$

which can be shown to have the expansion coefficients

$$c_n = 2^{-|n|}$$



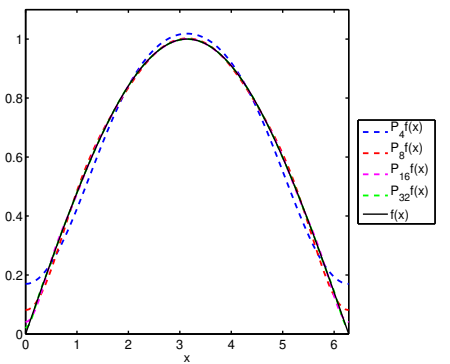
## Convergence of Fourier series

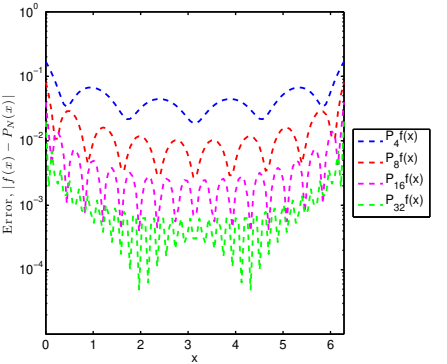
Consider the approximation of the $C_p^0([0, 2\pi])$ function

$$f_4(x) = \sin\left(\frac{x}{2}\right)$$

which can be shown to have the expansion coefficients

$$c_n = \frac{2}{\pi}\frac{1}{(1-4n^2)}$$



## Convergence of Fourier series

The complex Fourier series of a periodic function $f(x)$ is defined as

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}, \quad c_n = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} dx$$

Using orthogonality of the basis functions

$$\int_0^{2\pi} e^{inx} e^{-ikx} dx = \begin{cases} 0 & , k \neq n \\ 2\pi & , k = n \end{cases}$$

we can find the classical result referred to as Parseval's formula

$$\int_0^{2\pi} |f(x)|^2 dx = \int_0^{2\pi} f(x)\overline{f(x)} dx = 2\pi \sum_{n=-\infty}^{\infty} |c_n|^2$$

Thus, if $f(x)$ has bounded total variation then also the sum of the squares of the Fourier expansion coefficients is bounded.

Thus, the truncated Fourier series converges in the $L^2$-norm as

$$||f(x) - \mathcal{P}_N f(x)||_{L^2[0,2\pi]} = ||\tau||_{L^2[0,2\pi]} \to 0 \quad \text{as} \quad N \to \infty$$

# Convergence of Fourier series

But how fast does the series converge?

Suppose $f(x)$ is periodic and has continuous derivatives of orders $p = 0, 1, ..., m-1$ and $f^{(m)}(x)$ is integrable. By applying integration by parts it follows that

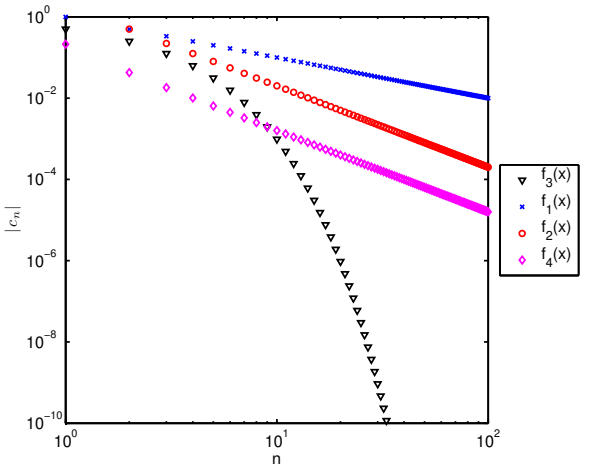
$$c_n = \frac{1}{2\pi(in)^m} \int_0^{2\pi} f^{(m)}(x) e^{-inx} dx$$

If $f^{(m)}(x)$ is integrable we find the upper bound

$$c_n \ll \frac{1}{n^m}, \quad n \to \infty$$

This implies that if $f(x)$ is smooth and periodic, the Fourier series converges rapidly and uniformly.

# Convergence rate of Fourier series




- Algebraic convergence if $c_n = \mathcal{O}\left(\frac{1}{n}^m\right)$ for $n \to \infty$.
- Spectral convergence if $c_n$ decay faster than algebraically!
  (think infinite order asymptotic convergence)
- If coefficients decay slowly, it requires a lot of work to evaluate $f(x)$ to high accuracy
  (may not be practical)

# Fourier series in practice

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}, \quad c_n = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} dx$$

Caveat's

- Easy to determine Fourier coefficients $c_n$ analytically for only a limited class of continuous functions.
- For arbitrary functions closed form expression for coefficients are not known.
- Point-wise convergence is determined by smoothness properties of $f(x)$.

We can try and resort to approximations...

# Discrete trigonometric polynomials

We know how to determine the coefficients of the continuous trigonometric polynomials, but

- How can we determine unknown coefficients for a discrete expansion?
- Can we retain the fast rate of convergence (accuracy) in approximate methods?
- How can it be done efficiently?

Main problem is orthogonal projection that requires evaluation of integrals.

What can we do?

## Discrete trigonometric polynomials

Consider the coefficients

$$c_n = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} dx$$

If we introduce a discrete set of $N+1$ points on the interval $x \in [0, 2\pi]$ (odd number of points)

$$x_j = jh, \quad h = \frac{2\pi}{N+1}, \quad j = 0, 1, ..., N$$

The integral can be approximated discretely, e.g., using the composite Trapezoidal (quadrature) rule, and we find the following approximation to the coefficients (odd expansion)

$$\tilde{c}_n = \frac{1}{N+1} \sum_{j=0}^{N} f(x_j) e^{-inx_j}$$

Now, how fast convergence do we get?

# Discrete trigonometric polynomials

If we use the discrete coefficients with an expansion for trigonometric polynomials with $N+1$ unknown coefficients (odd)

$$f(x) \approx \sum_{n=-N/2}^{N/2} \tilde{c}_n e^{inx}$$

Insert the expression for the discrete coefficients and we find

$$f(x) \approx \sum_{n=-N/2}^{N/2} \left( \frac{1}{N+1} \sum_{j=0}^{N} f(x_j) e^{-inx_j} \right) e^{inx}$$

If we change the order of summation we find

$$f(x) \approx \sum_{j=0}^{N} f(x_j) h_j(x), \quad h_j(x) = \frac{1}{N+1} \sum_{n=-N/2}^{N/2} e^{-inx_j} e^{inx}$$

## Discrete orthogonality

If we evaluate $h_j(x)$ at the discrete set of nodes

$$x_i = \tfrac{2\pi}{N+1} i, \quad i = 0, 1, ..., N$$

it is possible to show by a change of index and by use of the summation formula (convert sum to a closed-form expression)

$$\sum_{n=0}^{M} q^n = \frac{q^{M+1} - 1}{q - 1}$$

that

$$h_j(x_i) = \left( \frac{1}{N+1} \sum_{n=-N/2}^{N/2} e^{in(x_i - x_j)} \right) = \delta_{ij}$$

This result implies that the discrete approximation to the function $f(x)$ is in fact the (odd) Fourier Interpolant satisfying

$$\mathcal{I}_N f(x_n) = f(x_n), \quad n = 0, 1, ..., N$$

# Fourier interpolation (odd expansion)

The odd Fourier Interpolant of a real function $f(x)$, $x \in [0, 2\pi]$ is defined as

$$\mathcal{I}_N f(x) = \sum_{k=-N/2}^{N/2} \tilde{c}_k e^{ikx}$$

and satisfies the interpolation conditions

$$\mathcal{I}_N f(x_n) = f(x_n), \quad n = 0, 1, ..., N$$

for $x_n = \frac{2\pi n}{N+1}$.

The discrete coefficients (different from continuous coefficients) can be computed using the discrete inner product

$$\tilde{c}_k = \frac{1}{N+1} \sum_{j=0}^{N} f_j e^{-ikx_j} = \frac{1}{2\pi} \left(f, e^{ikx}\right)_N, \quad k = -N/2, ..., N/2$$

## Fourier interpolation (even expansion)

The even Fourier Interpolant of a real function $f(x)$, $x \in [0, 2\pi]$ is defined as

$$\mathcal{I}_N f(x) = \sum_{k=-N/2}^{N/2} \frac{\tilde{f}_k}{\bar{c}_k} e^{ikx}$$

where

$$\bar{c}_k = \begin{cases} 1, & k = -N/2+1, ..., N/2-1 \\ 2, & k = \pm N/2 \end{cases}$$

and satisfies the interpolation conditions

$$\mathcal{I}_N f(x_n) = f(x_n), \quad n = 0, 1, ..., N-1$$

for $x_n = \frac{2\pi n}{N}$.

The discrete coefficients are computed as

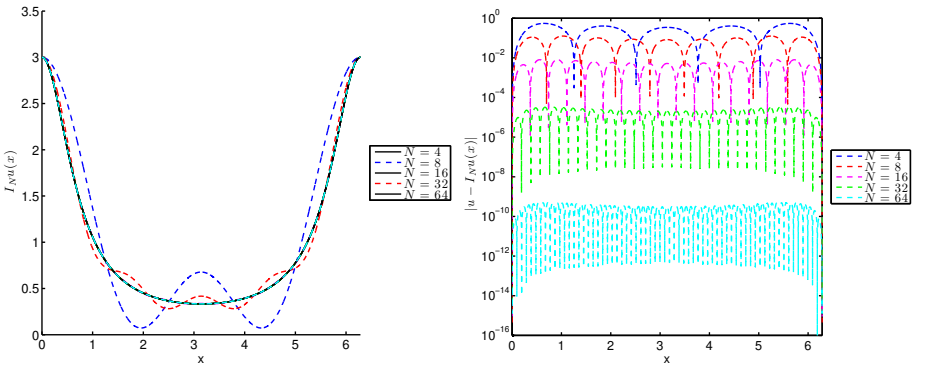
$$\tilde{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-ikx_j} = \frac{1}{2\pi} \left(f, e^{ikx}\right)_N, \quad k = -N/2, ..., N/2$$

## Fourier interpolation

Consider the approximation of the $C_p^\infty([0, 2\pi])$ function

$$f_3(x) = \frac{3}{5 - 4\cos(x)}$$

using a trigonometric polynomials based on coefficients computed using the Discrete Fourier Transform (DFT) method.



# Fourier interpolation (odd expansion)

To rewrite the Lagrange interpolating polynomial function into a closed form expression that is easy to evaluate, we use the trigonometric sum formula

$$\sum_{k=-K}^{K} e^{iks} = \frac{\sin\left((K+\frac{1}{2})s\right)}{\sin\left(\frac{1}{2}s\right)}$$

which can be derived from the partial summation formula

$$\sum_{n=0}^{M} q^n = \frac{q^{M+1}-1}{q-1}$$

We find

$$h_j(x) = \frac{1}{N+1}\frac{\sin\left(\frac{N+1}{2}(x-x_j)\right)}{\sin\left(\frac{1}{2}(x-x_j)\right)}, \quad j = 0, 1, ..., N$$

which is valid for a total of $N+1$ points $x_j = \frac{2\pi}{N+1}j$.

# Fourier interpolation (odd expansion)

The interpolation property of the trigonometric polynomials is a consequence of the orthogonal properties of the DFT.

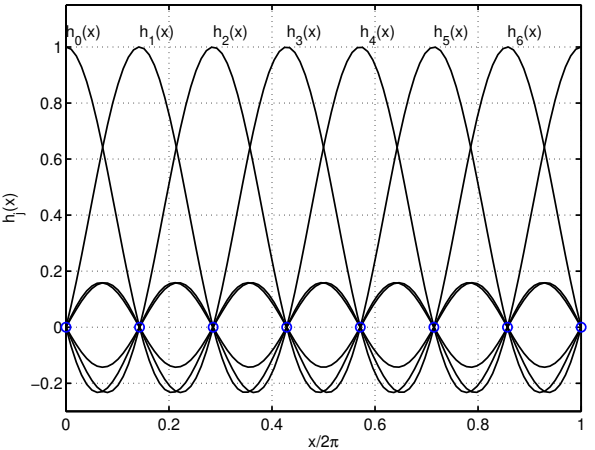



Illustration of Fourier interpolants for $N = 6$.

# Fourier interpolation (odd expansion)

Thus, the solution can be represented in Lagrange form as

$$\mathcal{I}_N u(x) = \sum_{j=0}^{N} u(x_j) h_j(x)$$

This form is referred to as a nodal expansion since the coefficients corresponds to the function values of $u(x)$ evaluated at the grid nodes.

# Discrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) is the mapping between the discrete set of complex values $f(x_j)$ and the complex coefficients $\tilde{c}_k$ of the trigonometric polynomial.

In summary, the $N$-point Discrete Fourier Transform (DFT) is defined as

$$\tilde{c}_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j) e^{-2\pi i j k/N}, \quad k = -N/2, .., N/2 - 1$$

and the inverse discrete Fourier Transform (IDFT) as

$$f(x_j) = \sum_{k=-N/2}^{N/2-1} \tilde{c}_k e^{2\pi i j k/N}, \quad j = 0, 1, ..., N-1$$

The DFT operations can be computed using the Fast Fourier Transform (FFT) method due to Cooley and Turkey (1967).

# Discrete Fourier Transform (DFT) in Matlab

The Discrete Fourier Transform (DFT) in Matlab is defined as

$$X_j = \sum_{k=1}^{N} x_k \exp\left[-ik\left(\tfrac{2\pi j}{N}\right)\right], \quad j = 1, 2, ..., N$$

where Matlab coefficients are normalized differently than the standard DFT. We have $\tilde{c}_j = \frac{1}{N} X_j$.

The inverse DFT is defined as

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \exp\left[in\left(\tfrac{2\pi k}{N}\right)\right], \quad n = 1, 2, ..., N$$

where $f(x_n) = N x_n$.

The actions of the DFT operations can be performed on a data set via the Matlab commands `fft` and `ifft`.

# Fast Fourier Transform (FFT) in Matlab[1]

The Fast Fourier Transform is defined on the interval $x \in [0, 2\pi[$ with $x_j = \frac{2\pi}{N} j$ and assumed periodic such that $x_0 = x_N$.

In Matlab the discrete Fourier coefficients produced by `fft` is stored according to the wave number space interval $[-N/2, N/2]$ in the following order

$$0, 1, ..., \tfrac{N}{2} - 1, -\tfrac{N}{2}, -\tfrac{N}{2} + 1, ..., -1$$

i.e. such that

$$\left[ \; \tilde{c}_0 \quad | \quad \tilde{c}_1 \quad \tilde{c}_2 \quad \cdots \quad \tilde{c}_{N/2-1} \quad | \quad \tilde{c}_{-N/2} \quad \tilde{c}_{N/2-1} \quad \cdots \quad \tilde{c}_{-1} \; \right]$$

- First half of coefficients corresponds to positive wavenumber parts.
- Second half of remaining coefficients corresponds to negative wavenumber parts.

[1] Check out: `http://www.fftw.org/`, the fastest Fourier transform in the West .

# Fast Fourier Transform (FFT) in Matlab

A couple of useful commands in Matlab

```
% reorder coefficients such that they follow
% natural orderinger from low to high wave numbers
>> N = 6;
>> fftshift([0:N-1])
ans =
   3     4     5     0     1     2

% undo fftshift ordering back to Matlab standard
>> ifftshift(fftshift([0:N-1]))
ans =
   0     1     2     3     4     5
```

# Zero-padding

Zero-padding is a technique that can be used for improving efficiency of DFT operations, interpolation via DFT and reducing aliasing errors in low wavenumber coefficients.

Assume the interpolating trigonometric polynomial of a function $f(x)$ is given on a $(N+1)$-point grid as

$$\mathcal{I}_N f(x) = \sum_{k=-N/2}^{N/2} \tilde{c}_k e^{ikx}$$

Then via zero-padding we can represent same trigonometric polynomial on a refined $(M+1)$-point grid $(M > N)$ as

$$\mathcal{I}_M f(x) = \mathcal{I}_N f(x) = \sum_{k=-M/2}^{M/2} \tilde{c}_k e^{ikx}$$

where $\tilde{c}_k = 0$ for $|k| > N/2$.

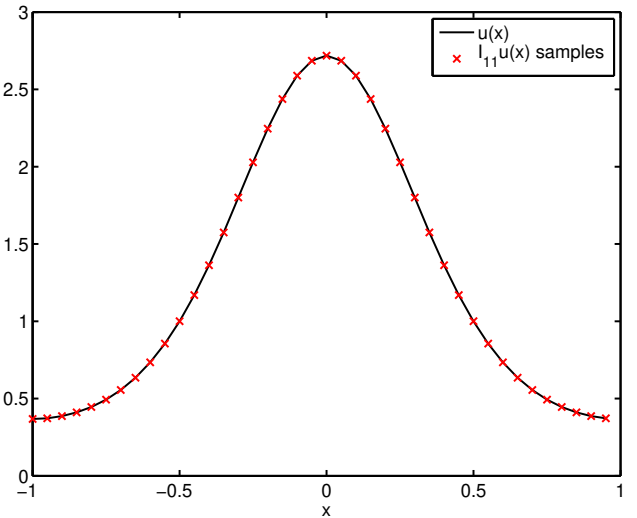
# Fast Fourier Transform (FFT) in Matlab

Matlab example of how **interpolation** can be done via FFT.

```
>> N  = 11;
>> x  = (0:N-1)/N*2-1;
>> u  = exp(cos(pi*x));
>> cn = fft(u)/N; % Normalization
>> cn = fftshift(cn);
>> M  = 40;
>> cm = zeros(1,M);
>> idx = round((M-1)/2+1+(-(N-1)/2:(N-1)/2))+mod(M,2);
>> cm(idx) = cn; % zero-padding
>> cm = fftshift(cm)*M; % Normalization
>> um = ifft(cm);
```

# Fast Fourier Transform (FFT) in Matlab

Example of interpolation of $I_{11}u(x)$ via zero-padding in Matlab with $u(x) = e^{\sin(x)}$ on $x \in [0, 2\pi[$ to a new grid $x_j$, $j = 0, 1, ..., 39$.



# Fast Fourier Transform (FFT) in Matlab

Matlab example of how **truncation** from a very fine grid solution can be done via FFT.

```
% Define very fine grid
>> MM = 16*N; % Modes
>> xx = (0:MM-1)/MM*2*pi; % very fine grid
% Interpolation
>> uu0 = fun(xx,t,M);
% Discrete fourier transform via FFT
>> uuhat0 = fft(uu0)/MM; % normalization
>> uuhat0 = fftshift(uuhat0); % reorder
% Index map for order N solution
>> idx = round((MM-1)/2+1+(-(N-1)/2:(N-1)/2))+mod(MM,2);
% Truncation to order N
>> uu1 = ifft( fftshift(uuhat0(idx))*N );
% Plotting
>> plot(x,u0,'k',xx,uu0,'r--',x,uu1,'bo'); xlabel('x')
>> legend('True','Interp. (alias.)','Proj. (de-alias.)')
```

# 1D Interpolation via FFT in Matlab

Matlab has a builtin function `interpft` that can be used to interpolate a signal efficiently via the FFT. The syntax is

```
>> y = interpft(x,n,dim)
```

and makes it possible to resample to $n$ equidistant nodes on same interval. Check the documentation.

Matlab example of how interpolation can be done via FFT.

```
>> N  = 11; x  = (0:N-1)/N*2-1;
>> u  = exp(cos(pi*x));
>> M = 5*N; y = (0:M-1)/M*2-1;
>> v  = interpft(u,M);
>> plot(x,u,'o',y,v,'x')
```

**Remark**: The number of interpolation nodes $M$ can be arbitrarily selected.

# Errors of Fourier Interpolation

We have found that interpolating property is satisfied when discrete Fourier coefficients is used for the construction of a trigonometric polynomial, i.e.

$$\mathcal{I}_N u(x_i) = u(x_i)$$

**Remark:** No error at interpolation nodes, however, with global basis functions there can be errors in between nodes.

The size of the error can be measured in the $L^2$-norm as

$$\begin{aligned} ||u(x) - \mathcal{I}_N u(x)||^2 &= \left|\left| \sum_{k=-N/2}^{N/2} (c_k - \tilde{c}_k) e^{ikx} + \sum_{|k|>N/2}^{\infty} c_k e^{ikx} \right|\right|^2 \\ &= 2\pi \left( \sum_{k=-N/2}^{N/2} |c_k - \tilde{c}_k|^2 + \sum_{|k|>N/2}^{\infty} |c_k|^2 \right) \end{aligned}$$

Thus, minimum occur when $\tilde{c}_k = c_k$ for $|k| \leq N/2$. Thus, interpolation errors $\geq$ truncation errors (due to aliasing).

## Approximation of derivatives

How can we approximate derivatives of functions?

If we have approximations based on partial sums of the form

$$\mathcal{P}_N u(x) = \sum_{n=0}^{N-1} c_n e^{ikx}$$

derivatives can be obtained by direct analytical differentiation

$$\frac{d^q}{dx^q}\mathcal{P}_N u(x) = \sum_{n=0}^{N-1} c_n (ik)^q e^{ikx}$$

We note the following

- With the Fourier coefficients $c_n$ known, the Fourier coefficients of the $q$'th derivative are given by $(ik)^q c_n$.
- The approximate expressions are of global nature and can be evaluated at arbitrary $x \in [0, 2\pi]$ with coefficients known.

# The Fourier Interpolation Derivative

If we choose to represent the solution in Lagrange form

$$\mathcal{I}_N u(x) = \sum_{j=0}^{N} u(x_j) h_j(x)$$

it is possible to determine derivatives of the $h_j(x)$ by algebraic differentiation using the rule

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

Direct differentiation of the interpolating polynomial leads to

$$\frac{d}{dx}\mathcal{I}_N u(x) = \sum_{j=0}^{N} u(x_j) h_j'(x)$$

# The Fourier Interpolation Derivative

If we determine $h_j'(x)$ and evaluate it at $x_k = \frac{2\pi}{N+1}k$, $k = 0, 1, ..., N$ we find

$$h_j'(x_k) = \frac{\frac{1}{2}(-1)^{k-l}}{\sin\left(\frac{\pi}{N+1}(k-l)\right)}$$

**Remark:** It shows that the Fourier Interpolation Derivative is identical to the infinite-order finite difference scheme!

From this derivative it also becomes clear that we can do differentiation via a matrix-vector product of the form

$$\frac{du_N}{dx} = Du_N, \quad D_{ij} = \begin{cases} \frac{1}{2}(-1)^{i+j}\left[\sin\left(\frac{\pi(i-j)}{N+1}\right)\right]^{-1} & , i \neq i \\ 0 & , i = j \end{cases}$$

with $D$ an odd Fourier derivative matrix for the first derivative of the interpolant evaluated at the grid nodes.

## Construction of derivative matrices

The derivative matrix $D$ can be precomputed and stored for use multiple times. The construction can be sensitive to rounding errors.

### Negative Sum Trick (NST)

Differentiation of a constant function should be zero. This implies that on a per grid point basis

$$\sum_{j=0}^{N} D_{ij} = 0, \quad i = 0, 1, ..., N$$

Enforce this condition explicitly by computing diagonal elements such that

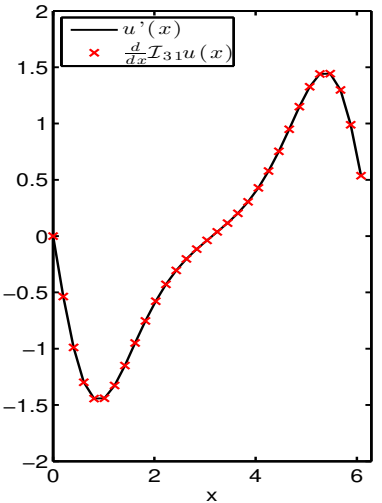
$$D_{ij} = - \sum_{j=0, j \neq i}^{N} D_{ij}$$

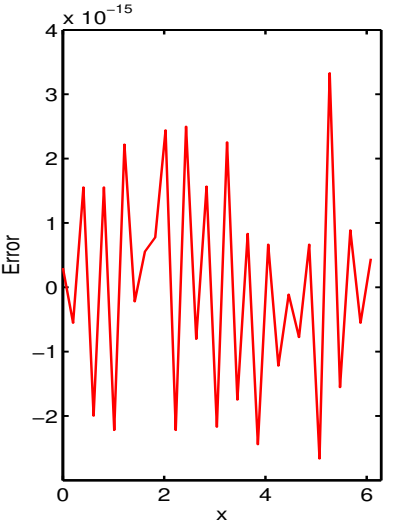
Using the NST it is some times possible to reduce rounding errors.

# Evaluation of derivative via FFT in Matlab

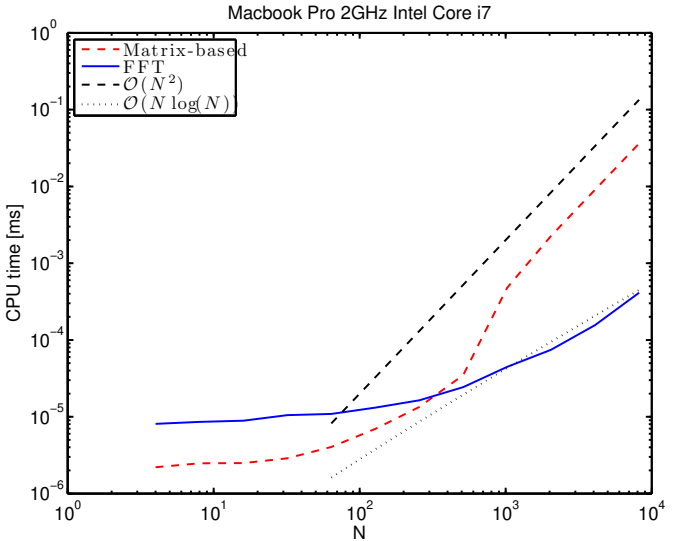
```
>> M  = 31;
% exact solution
>> fun  = @(x) exp(cos(x));
>> funx = @(x) -sin(x).*fun(x);
% range of wave numbers
>> kk = [0:round(M/2-1) -round(M/2-1):-1]
>> kk = kk*sqrt(-1);
% mesh paramters
>> h = 2*pi/M; x = (0:M-1)*h;
% Differentiation via FFT of periodic function.
>> u = fun(x);
>> ux = ifft(kk.*fft(u));
```

# Evaluation of derivative via FFT in Matlab



# Computing a first derivative in Matlab




**Note:** optimal performance is hardware and implementation dependent (see `fftw`).

# Phew!