July 20, 2013

Chapter 14: Numerical Differentiation

Uri M. Ascher and Chen Greif
Department of Computer Science
The University of British Columbia
{ascher,greif}@cs.ubc.ca

Slides for the book
A First Course in Numerical Methods (published by SIAM, 2011)
`http://www.ec-securehost.com/SIAM/CS07.html`

Goals of this chapter

- To develop useful formulas for approximating derivatives of a function $f(x)$ at a point $x = x_0$;
- to understand the mild stability limitations of numerical differentiation;
- *to see differentiation techniques in action in some more advanced applications.

Outline

- Deriving formulas using Taylor series
- Richardson extrapolation
- Deriving formulas using polynomial interpolation
- Roundoff and data errors
- *Differentiation matrices

*advanced

What is numerical differentiation

- Given a function $f(x)$ that is differentiable in the vicinity of a point x_0, it is often necessary to estimate the derivative $f'(x)$ and higher derivatives using nearby values of f.
- Example 1.2 in Chapter 1 provides a simple instance of numerical differentiation. Here we consider the more complete picture. For instance, we ask
  - how to achieve more, higher order difference formulas in an easy and orderly fashion?
  - how to control or altogether avoid the strong cancellation error effect demonstrated in Example 1.3?

  These and several other questions are considered here.

Why numerical differentiation?

- Numerical differentiation is a major tool in deriving methods for differential equations (see Chapter 16).
- Approximating derivatives is ubiquitous in continuous optimization and nonlinear equations (see Chapters 3 and 9).
- The need to estimate derivatives from discrete data often arises in applications.

Outline

- Deriving formulas using Taylor series
- Richardson extrapolation
- Deriving formulas using polynomial interpolation
- Roundoff and data errors
- *Differentiation matrices

Difference formulas using Taylor series

- This is the most convenient, ad hoc approach.
- Start from Taylor's expansion, generally written for a small $h > 0$ as

$$\begin{aligned} f(x_0 \pm h) &= f(x_0) \pm hf'(x_0) + \frac{h^2}{2}f''(x_0) \pm \frac{h^3}{6}f'''(x_0) + \\ &+ \frac{h^4}{24}f^{(iv)}(x_0) \pm \frac{h^5}{120}f^{(v)}(x_0) + \frac{h^6}{720}f^{(vi)}(x_0) + \mathcal{O}(h^7). \end{aligned}$$

- Truncate this as needed and derive an expression for $f'(x_0)$.
- Simplest example is the forward difference of Example 1.2. Likewise, backward difference is obtained by writing $f(x_0 - h) = f(x_0) - hf'(x_0) + \frac{h^2}{2}f''(\xi)$, hence $f'(x_0)$ is approximated by $\frac{f(x_0)-f(x_0-h)}{h}$ with truncation error $\frac{h}{2}f''(\xi)$ for some $x_0 - h \le \xi \le x_0$.
- The forward and backward formulas are one-sided, two-point formulas with truncation error $\mathcal{O}(h)$, i.e., they are 1st order methods.

Higher order formulas

- We can easily derive 2nd order, three-point formulas.
  - Centered: subtract expressions for $f(x_0+h)$ and $f(x_0-h)$:

$$f(x_0+h)-f(x_0-h)=2hf'(x_0)+\frac{2h^3}{6}f'''(x_0)+\frac{2h^5}{120}f^{(v)}(x_0)+\mathcal{O}(h^7).$$

    Thus, $f'(x_0)$ is approximated by $\frac{f(x_0+h)-f(x_0-h)}{2h}$ with truncation error $-\frac{h^2}{6}f'''(\xi)$.
  - One-sided: Please verify that the approximation $\frac{1}{2h}\left(-3f(x_0)+4f(x_0+h)-f(x_0+2h)\right)$ has truncation error $\frac{h^2}{3}f'''(\xi)$ where $\xi\in[x_0,x_0+2h]$.
- Using five points we can derive 4th order formulas, e.g., the formula

$$f'(x_0)\approx\frac{1}{12h}\left(f(x_0-2h)-8f(x_0-h)+8f(x_0+h)-f(x_0+2h)\right),$$

  has the truncation error $e(h)=\frac{h^4}{30}f^{(v)}(\xi)$.
- A similar approach may be used to eliminate for the 2nd derivative. For instance, the famous centred, three-point, 2nd order formula is

$$f''(x_0)=\frac{1}{h^2}\left(f(x_0-h)-2f(x_0)+f(x_0+h)\right)-\frac{h^2}{12}f^{(iv)}(\xi).$$

Higher order formulas

- We can easily derive 2nd order, three-point formulas.
  - Centered: subtract expressions for $f(x_0+h)$ and $f(x_0-h)$:

$$f(x_0+h)-f(x_0-h)=2hf'(x_0)+\frac{2h^3}{6}f'''(x_0)+\frac{2h^5}{120}f^{(v)}(x_0)+\mathcal{O}(h^7).$$

    Thus, $f'(x_0)$ is approximated by $\frac{f(x_0+h)-f(x_0-h)}{2h}$ with truncation error $-\frac{h^2}{6}f'''(\xi)$.
  - One-sided: Please verify that the approximation $\frac{1}{2h}\left(-3f(x_0)+4f(x_0+h)-f(x_0+2h)\right)$ has truncation error $\frac{h^2}{3}f'''(\xi)$ where $\xi\in[x_0,x_0+2h]$.
- Using five points we can derive 4th order formulas, e.g., the formula

$$f'(x_0)\approx\frac{1}{12h}\left(f(x_0-2h)-8f(x_0-h)+8f(x_0+h)-f(x_0+2h)\right),$$

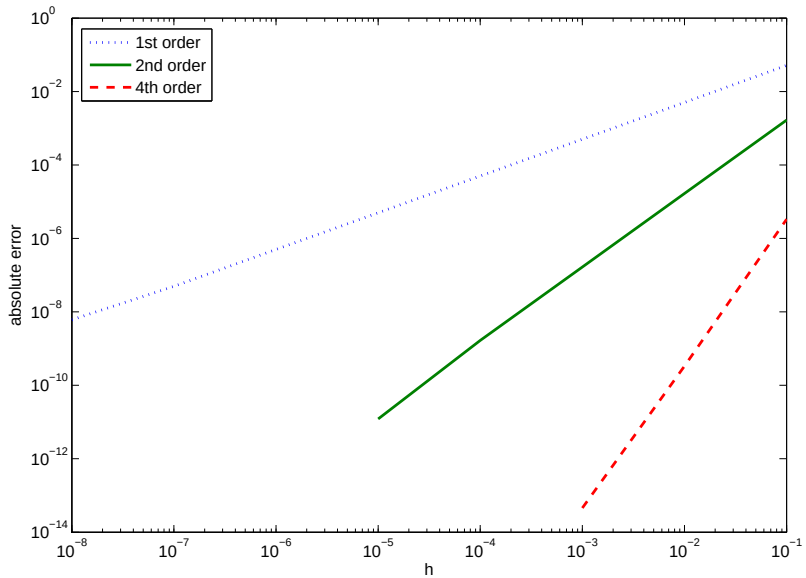
  has the truncation error $e(h)=\frac{h^4}{30}f^{(v)}(\xi)$.
- A similar approach may be used to eliminate for the 2nd derivative. For instance, the famous centred, three-point, 2nd order formula is

$$f''(x_0)=\frac{1}{h^2}\left(f(x_0-h)-2f(x_0)+f(x_0+h)\right)-\frac{h^2}{12}f^{(iv)}(\xi).$$

Example

Approximate $f'(0)$ for $f(x) = e^x$ using methods of order 1, 2 and 4. Record and plot errors for $h = 10^{-k},\ k = 1, \ldots, 5$.

h	$\frac{e^h - 1}{h} - 1$	$\frac{e^h - e^{-h}}{2h} - 1$	$\frac{-e^{2h} + 8e^h - 8e^{-h} + e^{-2h}}{12h} - 1$
0.1	5.17e-2	1.67e-3	3.33e-6
0.01	5.02e-3	1.67e-5	3.33e-10
0.001	5.0e-4	1.67e-7	−4.54e-14
0.0001	5.0e-5	1.67e-9	−2.60e-13
0.00001	5.0e-6	1.21e-11	−3.63e-12



Difference formulas

Notation: $f_j = f(x_0 + jh),\ j = 0, \pm 1, \pm 2, \ldots$.

Derivative	points	type	order	formula
$hf'(x_0)$	2	forward	1	$f_1 - f_0$
	2	backward	1	$f_0 - f_{-1}$
$2hf'(x_0)$	3	centred	2	$f_1 - f_{-1}$
	3	forward	2	$-3f_0 + 4f_1 - f_2$
$12hf'(x_0)$	5	centred	4	$f_{-2} - 8f_{-1} + 8f_1 - f_2$
$h^2 f''(x_0)$	3	centred	2	$f_{-1} - 2f_0 + f_1$
$12h^2 f''(x_0)$	5	centred	4	$-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2$

Taylor series approach assessment

- Simple and natural to use.
- Directly obtain both formula and its truncation error estimate.
- However, this approach is ad hoc and does not automatically generalize:
  - Sometimes we need to approximate with a non-uniform step size, e.g., approximate $f'(x_0)$ using $f(x_0),\ f(x_0+h),\ f(x_0-h/3)$. This would require an individual treatment.
  - Tedious work (and increased chances for human error) may be required for deriving high order formulas.

Outline

- Deriving formulas using Taylor series
- Richardson extrapolation
- Deriving formulas using polynomial interpolation
- Roundoff and data errors
- *Differentiation matrices

Richardson extrapolation

- This is a straightforward, favourite technique based on a simple yet general principle for deriving higher order formulas from lower order ones.
- More limited than the general Taylor series approach, but methodical and easily applied.
- Can be applied repeatedly for generating methods of higher and higher order.
- Used in classical methods for numerical integration and differential equations (Chapters 15, 16).
- **Basic idea**: use lower order formula with more than one step size, and combine to eliminate the leading term of the truncation error.

Example of method derivation

Goal: Derive a centred, 4th order formula for the 2nd derivative.

- Write the centred 3-point formula for f_0 once for h and once for $2h$,

$$\begin{aligned} f''(x_0) &= \frac{1}{h^2}(f_{-1} - 2f_0 + f_1) - \frac{h^2}{12}f^{(iv)}(x_0) + \mathcal{O}(h^4) \\ &= \frac{1}{(2h)^2}(f_{-2} - 2f_0 + f_2) - \frac{(2h)^2}{12}f^{(iv)}(x_0) + \mathcal{O}(h^4). \end{aligned}$$

- The leading term of the truncation error in the second line is 4 times larger than in the first. So multiply the first line by 4 and subtract the 2nd line:

$$3f''(x_0) = \frac{4}{h^2}(f_{-1} - 2f_0 + f_1) - \frac{1}{(2h)^2}(f_{-2} - 2f_0 + f_2) + \mathcal{O}(h^4).$$

- Rearrange:

$$f''(x_0) = \frac{1}{12h^2}(-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2) + \mathcal{O}(h^4).$$

Outline

- Deriving formulas using Taylor series
- Richardson extrapolation
- Deriving formulas using polynomial interpolation
- Roundoff and data errors
- *Differentiation matrices

Deriving methods using polynomial interpolation

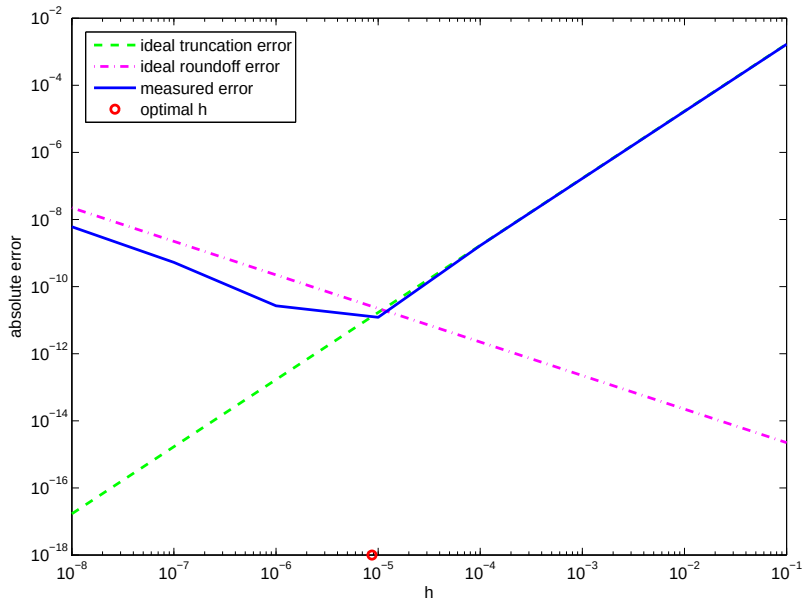
- This is a natural approach for Chapter 10 veterans: simply pass a polynomial of degree n, interpolating f and x_0 plus n of its neighbours; differentiate the polynomial; and evaluate at x_0.
- Obtain a general formula for the first derivative.
- Approach handles also non-uniform step sizes (points can be arbitrarily spaced).
- Immediate corollary: for deriving a method of order r we need at least $r+1$ points! (Methods which require *only* $r+1$ points are called compact.)
- Very useful for specific purposes such as Chebyshev differentiation matrix (next section).
- However, the resulting formulas, while general, can also be complex-looking and ugly. The Taylor series approach, while more ad hoc, is often more natural to use.

Outline

- Deriving formulas using Taylor series
- Richardson extrapolation
- Deriving formulas using polynomial interpolation
- Roundoff and data errors
- *Differentiation matrices

Roundoff and data errors

- Recall Example 1.3: for very small h and smooth f we may encounter severe cancellation error amplified by h^{-1} when approximating the derivative f'. Then, roundoff error takes over, and it grows as h decreases.
- Clearly, this problem happens with all numerical differentiation methods we have seen thus far (e.g. see our table of difference formulas).
- In fact, for higher order methods roundoff error dominates sooner (i.e. for larger h) because truncation error decreases faster.



Example

Consider the 2nd order method $f'(x_0) \simeq D_h = \frac{f(x_0+h)-f(x_0-h)}{2h}$.

- Denote $\mathrm{fl}(f(x)) \equiv \overline{f}(x) = f(x) + e_r(x), \quad |e_r(x)| \le \epsilon$, where ϵ depends on the **rounding unit**. Assuming exact arithmetic for simplicity, $\overline{D}_h = \frac{\overline{f}(x_0+h)-\overline{f}(x_0-h)}{2h}$.
- Obtain

$$\begin{aligned} |\overline{D}_h - D_h| &= \left| \frac{\overline{f}(x_0+h) - \overline{f}(x_0-h)}{2h} - \frac{f(x_0+h) - f(x_0-h)}{2h} \right| \\ &= \left| \frac{e_r(x_0+h) - e_r(x_0-h)}{2h} \right| \\ &\le \left| \frac{e_r(x_0+h)}{2h} \right| + \left| \frac{e_r(x_0-h)}{2h} \right| \le \frac{\epsilon}{h}. \end{aligned}$$

- So, if $|f'''(\xi)| \le M$ then

$$\begin{aligned} \left|f'(x_0) - \overline{D}_h\right| &= \left|(f'(x_0) - D_h) + \left(D_h - \overline{D}_h\right)\right| \\ &\le |f'(x_0) - D_h| + \left|D_h - \overline{D}_h\right| \le \frac{h^2 M}{6} + \frac{\epsilon}{h}. \end{aligned}$$

- "Theoretically optimal" h is where this bound is minimized: $h_* = (3\epsilon/M)^{1/3}$.

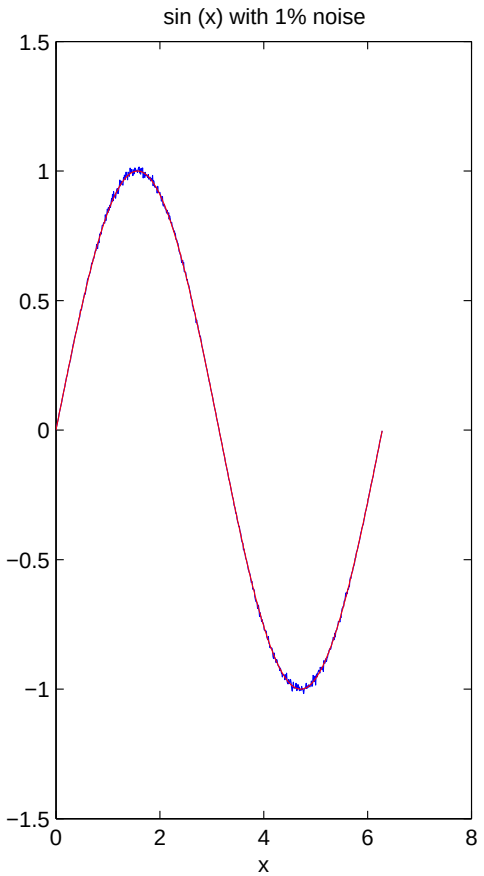
How bad can this problem be, and what can we do?

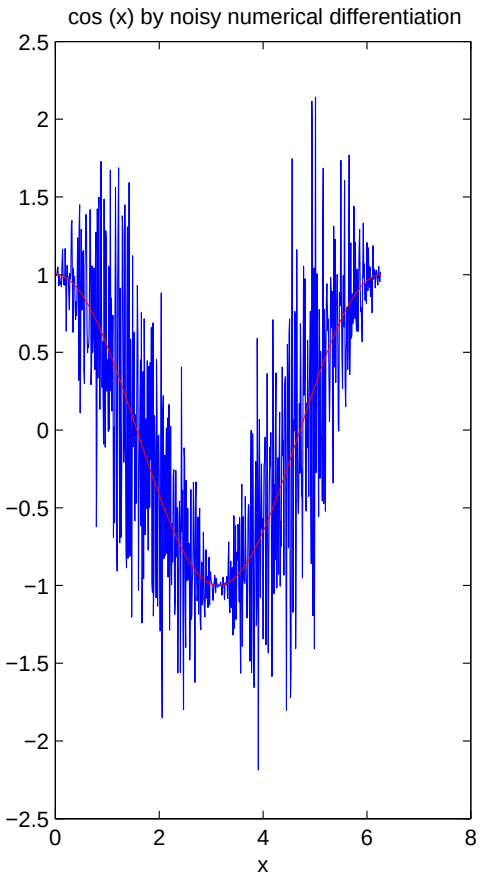
- The answer highly depends on the application!
- For discretization of differential equations, can usually keep $h \gg \epsilon$ using IEEE standard "double precision" word (but not "single precision"); see Chapter 2.
- For calculating gradient and Hessian in a more controlled optimization context (also, constrained multibody simulations), it is usually fine to use h "not too small". An alternative is automatic differentiation methods, not covered here, which do not suffer from roundoff error problems.
- For applications involving numerical differentiation of measured data which may contain noise, it is easy to get stuck! A possible remedy is to smooth the data (i.e., approximate the data by a smooth function), and only then differentiate.

Example: differentiating noisy data

We create a function $f(x)$ to be differentiated by first sampling $\sin(x)$ on $[0, 2\pi]$ with $h = .01$, and then adding 1% Gaussian noise to these 200π values. The result is in the left panel below.

Next, numerical differentiation approximately gives $\cos(x)$ plus the noise magnified by $1/h = 100$. The (unacceptable) result is depicted in the right panel below.





Outline

- Deriving formulas using Taylor series
- Richardson extrapolation
- Deriving formulas using polynomial interpolation
- Roundoff and data errors
- *Differentiation matrices

Differentiation matrices

- Function differentiation at a point x is a local process, and our formulas use this to generate **local** derivative approximations.
- However, in many applications the differentiation process is applied for each point of an entire mesh: a sequence of points $x_0 < x_1 < \cdots < x_{n-1} < x_n$ which represents a discretization of an interval (see Chapters 11, 13, 16).
- Upon applying a numerical differentiation formula to each point of the mesh, a matrix operator is obtained.
- We'll see four differentiation matrices next: (i) centred differences, (ii) one-sided differences, (iii) polynomial interpolation at Chebyshev extremum points, and (iv) Fourier differentiation matrix.

Finite differences: centered

- Centered difference: $\mathbf{f}' = D\mathbf{f}$ where

$$\mathbf{f}' \approx \begin{pmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_{n-1}) \end{pmatrix}, D = \frac{1}{2h} \begin{pmatrix} -1 & 0 & 1 & & \\ & -1 & 0 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 0 & 1 \end{pmatrix}, \mathbf{f} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{pmatrix}.$$

  The $(n-1) \times (n+1)$ matrix D is a simple instance of a differentiation matrix.

- However, D is not square, let alone invertible.

Finite differences: centered

- Centered difference: $\mathbf{f}' = D\mathbf{f}$ where

$$\mathbf{f}' \approx \begin{pmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_{n-1}) \end{pmatrix}, D = \frac{1}{2h} \begin{pmatrix} -1 & 0 & 1 & & \\ & -1 & 0 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 0 & 1 \end{pmatrix}, \mathbf{f} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{pmatrix}.$$

  The $(n-1) \times (n+1)$ matrix D is a simple instance of a differentiation matrix.

- However, D is not square, let alone invertible.

Finite differences: one-sided

- Assuming also $f(x_0) = 0$, can get a square, invertible D upon using a one-sided formula:

$$\begin{pmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_n) \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix},$$

  where the first row in the matrix corresponds to the initial condition $f_0 = f(x_0) = 0$.

- However, there is a bit of "cheating" involved here, as this is not a pure differentiation process and the assumption $f_0 = 0$ is arbitrary.
- On the other hand, interpolating mesh values by a polynomial and subsequently differentiating it naturally leads to a proper, square differentiation matrix!

Finite differences: one-sided

- Assuming also $f(x_0) = 0$, can get a square, invertible D upon using a one-sided formula:

$$\begin{pmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_n) \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix},$$

  where the first row in the matrix corresponds to the initial condition $f_0 = f(x_0) = 0$.

- However, there is a bit of "cheating" involved here, as this is not a pure differentiation process and the assumption $f_0 = 0$ is arbitrary.
- On the other hand, interpolating mesh values by a polynomial and subsequently differentiating it naturally leads to a proper, square differentiation matrix!

Finite differences: one-sided

- Assuming also $f(x_0) = 0$, can get a square, invertible D upon using a one-sided formula:

$$\begin{pmatrix} f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_n) \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix},$$

  where the first row in the matrix corresponds to the initial condition $f_0 = f(x_0) = 0$.

- However, there is a bit of "cheating" involved here, as this is not a pure differentiation process and the assumption $f_0 = 0$ is arbitrary.
- On the other hand, interpolating mesh values by a polynomial and subsequently differentiating it naturally leads to a proper, square differentiation matrix!

Differentiating Chebyshev interpolant at extremum points

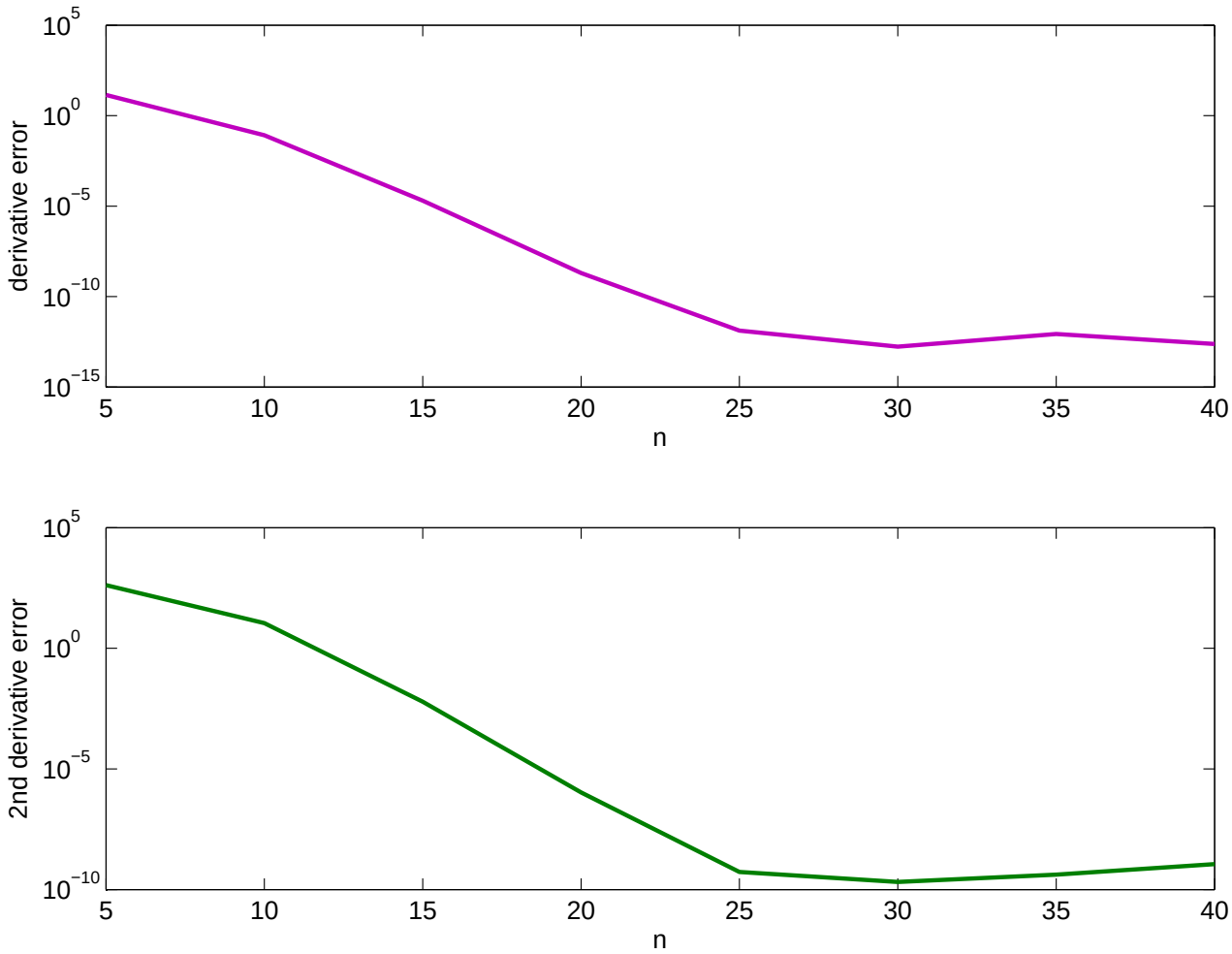
- Construct a polynomial interpolant, differentiate it and evaluate at the same interpolation abscissae.
- Choose the extremum points of the Chebyshev polynomial (Section 12.4)

$$x_i = \cos\left(\frac{i}{n}\pi\right), \quad i = 0, \ldots, n.$$

  Denote result $D_C\mathbf{f}$.
- Use Lagrange interpolation form (Section 10.3) because n may be large and maintaining stability is important.
- Can obtain highly accurate derivative values this way, without restrictions such as periodicity.

Example

$f(x) = e^x \sin(10x)$ on $[0, \pi]$.



Fourier differentiation matrix

- For equally spaced function values (see Section 13.3)

$$\begin{aligned} c_k &= \frac{1}{m}\sum_{i=0}^{m-1} y_i e^{-\imath k x_i}, \quad -l \le k \le l-1, \\ p_n(x) &= \sum_{k=-l}^{l-1} c_k e^{\imath k x}, \quad \text{hence} \\ y'_i &= p'_n(x_i) = \sum_{k=-l}^{l-1} \imath k c_k e^{\imath k x_i}, \quad i = 0, 1, \ldots, m-1. \end{aligned}$$

- Hence differentiation matrix is diagonalized!
- Use FFT, of course:

$$D_F = \mathcal{F}^{-1}\left[\imath \operatorname{diag}\{-l, -l+1, \ldots, l-1\}\right]\mathcal{F}.$$

In Matlab, FFT for rth derivative

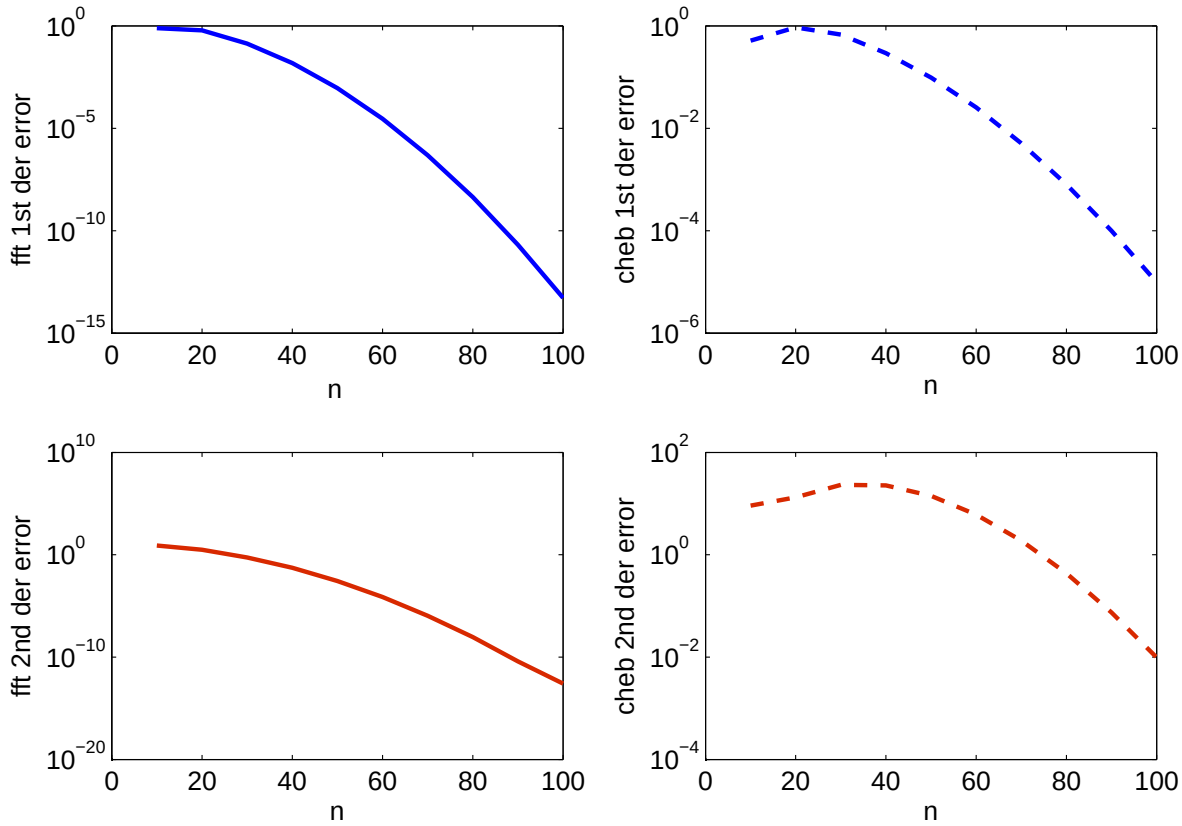
Assume f is column array containing the values $f(x_i)$ on $[a, b]$
Let $m2 = 0$ if r is odd, $m2 = l$ otherwise. Use instructions:

$$
\begin{aligned}
&fhat = fft(f);\\
&ghat = (i)^r * ([0 : l - 1 m2 - l + 1 : -1].^r)'. * fhat;\\
&ghat = (2 * pi/(b - a))^r * ghat;\\
&g = real(ifft(ghat));
\end{aligned}
$$

Obtain excellent accuracy if $f(x)$ is (almost) periodic on $[a, b]$.

Example

$f(x) = e^{-5x^2}$ on $[-2\pi, 2\pi]$.



Here Fourier is much better than Chebyshev. (But for functions that are "far from periodic" the opposite is true.)

Solving a partial differential equation (PDE): splitting cubic NLS

Schrödinger equation with a cubic nonlinearity

$$\psi_t = \imath(\psi_{xx} + |\psi|^2\psi).$$

- Here f_t is 1st derivative of f wrto time t, f_{xx} is 2nd derivative of f wrto space x.
- Assume periodic boundary conditions (BC) on ψ.
- Norm preservation $\|\psi(t)\|^2 = \|\psi(0)\|^2,\ \forall t$
- Hamiltonian PDE $H(\psi, \bar{\psi}) = \psi_x\bar{\psi}_x - \frac{1}{2}\psi^2\bar{\psi}^2$: the spatial integral of this is preserved in time.

Solving a partial differential equation (PDE): splitting cubic NLS

Schrödinger equation with a cubic nonlinearity

$$\psi_t = \imath(\psi_{xx} + |\psi|^2\psi).$$

- Here f_t is 1st derivative of f wrto time t, f_{xx} is 2nd derivative of f wrto space x.
- Assume periodic boundary conditions (BC) on ψ.
- Norm preservation $\|\psi(t)\|^2 = \|\psi(0)\|^2, \ \forall t$
- Hamiltonian PDE $H(\psi, \bar{\psi}) = \psi_x \bar{\psi}_x - \frac{1}{2}\psi^2\bar{\psi}^2$: the spatial integral of this is preserved in time.

Solving a partial differential equation (PDE): splitting cubic NLS

Schrödinger equation with a cubic nonlinearity

$$\psi_t = \imath(\psi_{xx} + |\psi|^2\psi).$$

- Here f_t is 1st derivative of f wrto time t, f_{xx} is 2nd derivative of f wrto space x.
- Assume periodic boundary conditions (BC) on ψ.
- Norm preservation $\|\psi(t)\|^2 = \|\psi(0)\|^2, \ \forall t$
- Hamiltonian PDE $H(\psi, \bar{\psi}) = \psi_x\bar{\psi}_x - \frac{1}{2}\psi^2\bar{\psi}^2$: the spatial integral of this is preserved in time.

A splitting method

- Split the PDE to a linear part and an ordinary DE (ODE) part:
  1. For the linear PDE

     $$u_t = \imath u_{xx},$$

     apply standard 3-point discretization in space and **implicit midpoint** in time.
  2. For the ODE

     $$u_t = \imath |u|^2 u,$$

     the exact ODE solution for each x is

     $$u(t + \Delta t) = u(t)\ e^{\imath \Delta t |u|^2}.$$

- Composition: at each time step $t = t_k$, with the current solution u^k:
  1. approximately solve the linear PDE problem from t_k to t_{k+1}, obtaining $\hat{u}$;
  2. solve the ODE at each mesh point $x = x_j$ from t_k to t_{k+1}, starting from $\hat{u}_j$ and calling the result u^{k+1}.

A splitting method

- Split the PDE to a linear part and an ordinary DE (ODE) part:
  1. For the linear PDE

     $$u_t = \imath u_{xx},$$

     apply standard 3-point discretization in space and **implicit midpoint** in time.
  2. For the ODE

     $$u_t = \imath |u|^2 u,$$

     the exact ODE solution for each x is

     $$u(t + \Delta t) = u(t)\ e^{\imath \Delta t |u|^2}.$$

- Composition: at each time step $t = t_k$, with the current solution u^k:
  1. approximately solve the linear PDE problem from t_k to t_{k+1}, obtaining $\hat{u}$;
  2. solve the ODE at each mesh point $x = x_j$ from t_k to t_{k+1}, starting from $\hat{u}_j$ and calling the result u^{k+1}.

A splitting method

- Split the PDE to a linear part and an ordinary DE (ODE) part:
  1. For the linear PDE

     $$u_t = \imath u_{xx},$$

     apply standard 3-point discretization in space and **implicit midpoint** in time.
  2. For the ODE

     $$u_t = \imath |u|^2 u,$$

     the exact ODE solution for each x is

     $$u(t + \Delta t) = u(t)\ e^{\imath \Delta t |u|^2}.$$

- Composition: at each time step $t = t_k$, with the current solution u^k:
  1. approximately solve the linear PDE problem from t_k to t_{k+1}, obtaining $\hat{u}$;
  2. solve the ODE at each mesh point $x = x_j$ from t_k to t_{k+1}, starting from $\hat{u}_j$ and calling the result u^{k+1}.

Example: solitons

Periodic BC on $[-20, 80]$. IC

$$\psi(0,x) = e^{\imath x/2}\text{sech}(x/\sqrt{2}) + e^{\imath(x-25)/20}\text{sech}((x-25)/\sqrt{2}).$$

t	Δt	Δx	Error-Ham	Error-norm
200	.1	.1	3.7e-5	4.3e-13
	.01	.01	3.9e-9	1.5e-11

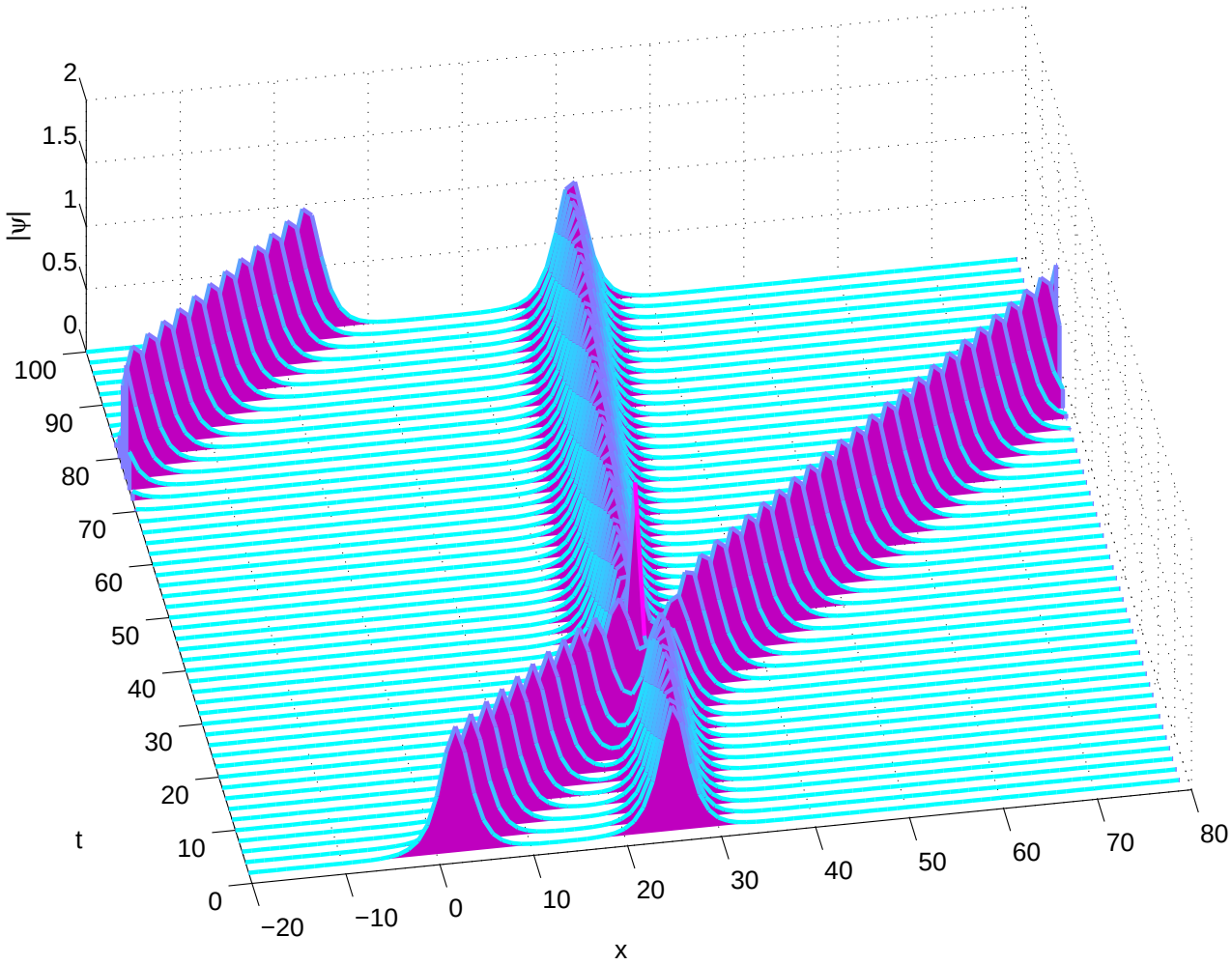
Example: solitons

Periodic BC on $[-20, 80]$. IC

$$\psi(0,x) = e^{\imath x/2}\text{sech}(x/\sqrt{2}) + e^{\imath(x-25)/20}\text{sech}((x-25)/\sqrt{2}).$$

t	Δt	Δx	Error-Ham	Error-norm
200	.1	.1	3.7e-5	4.3e-13
	.01	.01	3.9e-9	1.5e-11

Soliton solution progress



Spectral splitting method

- Another method: same splitting of the differential equation to a linear PDE plus an ODE, but instead of the finite difference discretizations for the derivatives, solve $u_t = \imath u_{xx}$ using a spectral method in both space and time:

$$u(t + \Delta t) = \mathcal{F}^{-1}\left(e^{-\imath\xi^2 \Delta t}\mathcal{F}(u(t))\right).$$

- Discretize: $u(t) \equiv u^n = (u_1^n, \ldots, u_J^n)$, $u_j^n \approx u(j\Delta x, n\Delta t)$, $u(t + \Delta t) \equiv u^{n+1}$, and $\mathcal{F}$ is the fast Fourier transform (FFT).
- This works well, producing beautiful, accurate solitons using a very short program. The error due to splitting, however, is generally not better than in the finite difference case.