EE 3911
數理特論：偏微分方程與數值方法
Partial Differential Equations and Numerical Methods

Ray-Kuang Lee*

Institute of Photonics Technologies,
Department of Electrical Engineering, and Department of Physics,
National Tsing-Hua University

*rklee@ee.nthu.edu.tw

Textbook, Reference books:

★ **Textbook:**

[T] Erwin Kreyszig, "Advanced Engineering Mathematics," 10th Ed. (John Wiley & Sons Inc., 2011).

Ch. 4, 5, 12, 19, 20, 21

★ **Reference books:**

- **[N] Ward Cheney and David Kincaid, "Numerical Mathematics and Computing," 6th Ed. (Brooks/Cole, 2008).**

  Ch. 1-5, 6, 7, 8, 10-11,12, 14, 15

- **[A] J.D. Logan, "Applied Mathematics," 3rd ed. (Wiley, 2006).**

  Ch. 1, 3, 4, 5, 6

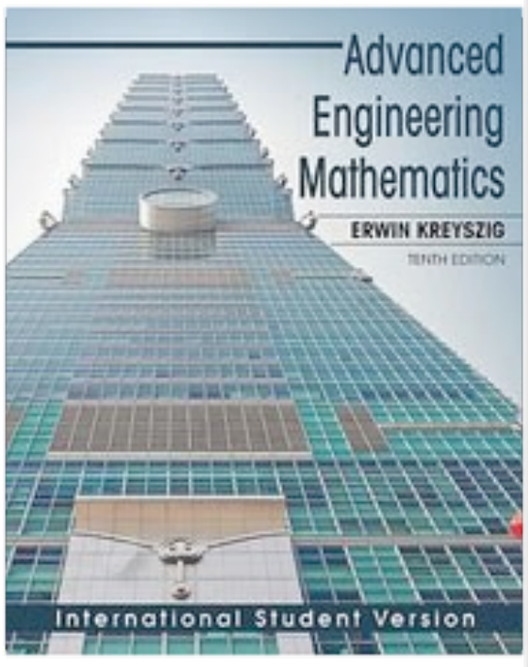
★ **Office hours:**

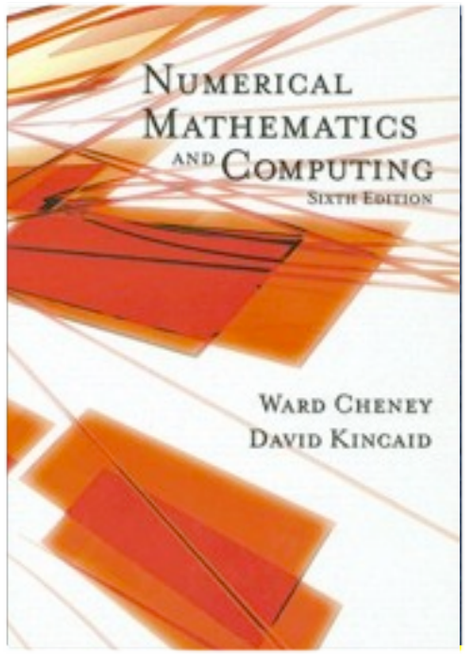
M78W78 at R523, EECS bldg.

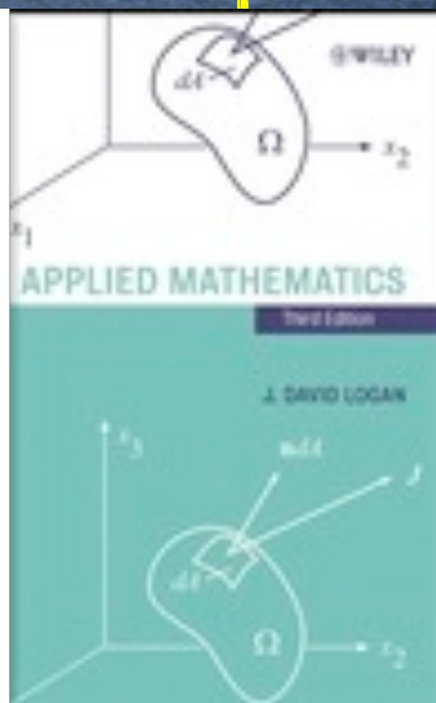
★ **TA: Johnny, 李政誼**

Email:

johnnyygod2002@hotmail.com







Syllabus (Spring 2012):

I. **Basics of Numerical Methods: 4 weeks** (~~2/28~~, 3/6, 3/13, 3/20, 3/27, ~~4/3~~)
   1. Floating-Point Representation and Errors, T19.1, N2
   2. Roots of Equations, T19.2, N3
   3. Interpolations, T19.3, T19.4, N4
   4. Numerical Differentiations, T19.5, N4
   5. Numerical Integrations, T19.5, N5
   6. Numerical Linear Algebra, T20, N7, N8
   7. Runge-Kutta methods for ODEs, T21.1, T21.2, T21.3, N10, N11

II. **Numerical Methods for PDEs: 5 weeks** (4/10, 4/17, 4/24, 5/1, 5/8)
   1. PDEs and Finite-Difference method, T12, N15, A6
   2. Crank-Nicolson method fro Parabolic problems, T21.6, N15.1
   3. Lax-Wendroff method for Hyperbolic problems, T21.7, N15.2
   4. Gauss-Seidel method for Elliptic Problems, T20.3, T21.4, T21.5, N15.3

Week 1: (3/6)

1. **Errors**: [T] Ch. 19.1; [C] Ch. 1, Ch. 2
   - Floating-Points representation,
   - Roundoff Errors,
   - Truncation Errors,
   - Error Propagation,
   - Dimensionless equation.

2. **Root of equations**: [T] Ch. 19.2; [C] Ch. 3
   - Bisection method,
   - Fixed-point iteration method,
   - Newton's method,
   - Newton's method for nonlinear systems.
   - Secant method.

Floating-Points representation

In floating-point representation,

$$s \times M \times B^{e-E},$$

- s: sign bit, $b_{63} = \pm$
- B: the base of the representation (usually $B = 2$, but sometimes $B = 16$),
- e: exact integer exponent, $(b_{62}b_{61}\cdots b_{52})$,
- E: the bias of the exponent, $2^{11-1} - 1 = 1023$

$$e - E = -1022 \backsim +1023$$

- M: exact positive integer mantissa,

$$\begin{aligned} M &= 0.b_{51}b_{50}\cdots b_1b_0 = [b_{51}b_{50}\cdots b_1b_0] \times 2^{-52}, \quad \text{un-normalized} \\ &= 1.b_{51}b_{50}\cdots b_1b_0 = 1 + [b_{51}b_{50}\cdots b_1b_0] \times 2^{-52}, \quad \text{normalized} \end{aligned}$$

Floating-Points representation

IEEE Single Precision: 32bit

$\pm$	$a_1 a_2 a_3 \ldots a_8$	$b_1 b_2 b_3 \ldots b_{23}$

If exponent bitstring $a_1 \ldots a_8$ is	Then numerical value represented is
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{-125}$
$(00000011)_2 = (3)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{-124}$
$\downarrow$	$\downarrow$
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{1}$
$\downarrow$	$\downarrow$
$(11111100)_2 = (252)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{125}$
$(11111101)_2 = (253)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{126}$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ if $b_1 = \ldots = b_{23} = 0$, NaN otherwise

Floating-Points representation

IEEE Double Precision: 64bit

$\pm$	$a_1 a_2 a_3 \ldots a_{11}$	$b_1 b_2 b_3 \ldots b_{52}$

If exponent bitstring is $a_1 \ldots a_{11}$	Then numerical value represented is
$(00000000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{-1022}$
$(00000000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{-1022}$
$(00000000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{-1021}$
$(00000000011)_2 = (3)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{-1020}$
$\downarrow$	$\downarrow$
$(01111111111)_2 = (1023)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{0}$
$(10000000000)_2 = (1024)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{1}$
$\downarrow$	$\downarrow$
$(11111111100)_2 = (2044)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{1021}$
$(11111111101)_2 = (2045)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{1022}$
$(11111111110)_2 = (2046)_{10}$	$\pm(1.b_1 b_2 b_3 \ldots b_{52})_2 \times 2^{1023}$
$(11111111111)_2 = (2047)_{10}$	$\pm\infty$ if $b_1 = \ldots = b_{52} = 0$, NaN otherwise

IEEE 64 bit Floating-Points

- Least Significant Digit:

$$\Delta M = 2^{-52} \approx 10^{-52*3/10} = 10^{-16}$$

- Least Significant Bit:

$$\Delta M \times 2^{e-E} = 2^{-52} \times 2^{e-E}$$

- minimum positive number:

$$(0 + 2^{-52}) \times 2^{-1022} = 2^{-1074} \approx 10^{-1074*3/10} = 10^{-321}$$

- maximum positive number:

$$(2 - 2^{-52}) \times 2^{1023} \approx 10^{308}$$

$2^{10} = 1024 \approx 10^3$

Errors:

- Round-off Error: by finite bits,
- Truncation Error: by finite number of terms (operations),
- Overflow/Underflow: by too large or too small numbers,
- Negligible Addition: by adding two numbers differing by over 52 bits, "+"
- Loss of significance: by a "bad subtraction", "−"
- Error Magnification: by multiplying/dividing a number containing a small error, "×" and "/"
- Errors by algorithms.

Loss of Significance:

Example:

$$
\begin{aligned}
f(x) &= \sqrt{x^2} - 1, && x = 0 \\
&= \frac{x^2}{\sqrt{x^2+1}+1}
\end{aligned}
$$

Example:

$$
\begin{aligned}
f(x) &= x - \sin x, && x = 0 \\
&= x - (x - \frac{x^3}{31} + \frac{x^5}{5!} + \ldots) \\
&= \frac{x^3}{31} - \frac{x^5}{5!} + \ldots
\end{aligned}
$$

Loss of Significance:

Example:

$$
\begin{aligned}
f(x) &= e^x - e^{-2x}, \qquad x = 0 \\
&= (1 + x + \frac{x^2}{21} + \ldots) - (1 - 2x + \frac{4x^2}{2!} - \ldots) \\
&= 3x - \frac{3}{2}x^2 + \frac{3}{2}x^3 + \ldots \\
\text{or} \quad &= e^{-2x}(e^{3x} - 1) \\
&= e^{-2x}(3x + \frac{9}{2!}x^2 + \ldots)
\end{aligned}
$$

Normalization:

Nonlinear Schrödinger equation:

$$i\frac{\partial U}{\partial z} + D_2\frac{\partial^2 U}{\partial t^2} + \chi_3|U|^2U = 0$$

- set $\eta = az$ and $\tau = bt$, then NLSE becomes

$$ia\frac{\partial U}{\partial \eta} + D_2b^2\frac{\partial^2 U}{\partial \tau^2} + \chi_3|U|^2U = 0$$

- By choosing

$$\begin{aligned} a &= \chi_3, \\ b &= \sqrt{\frac{\chi_3}{2D_2}}, \end{aligned}$$

NLSE is reduced to scaleless (dimensionless) form,

$$i\frac{\partial U}{\partial \eta} + \frac{1}{2}\frac{\partial^2 U}{\partial \tau^2} + |U|^2U = 0.$$

Roots of equation:

$$F(x) = 0$$

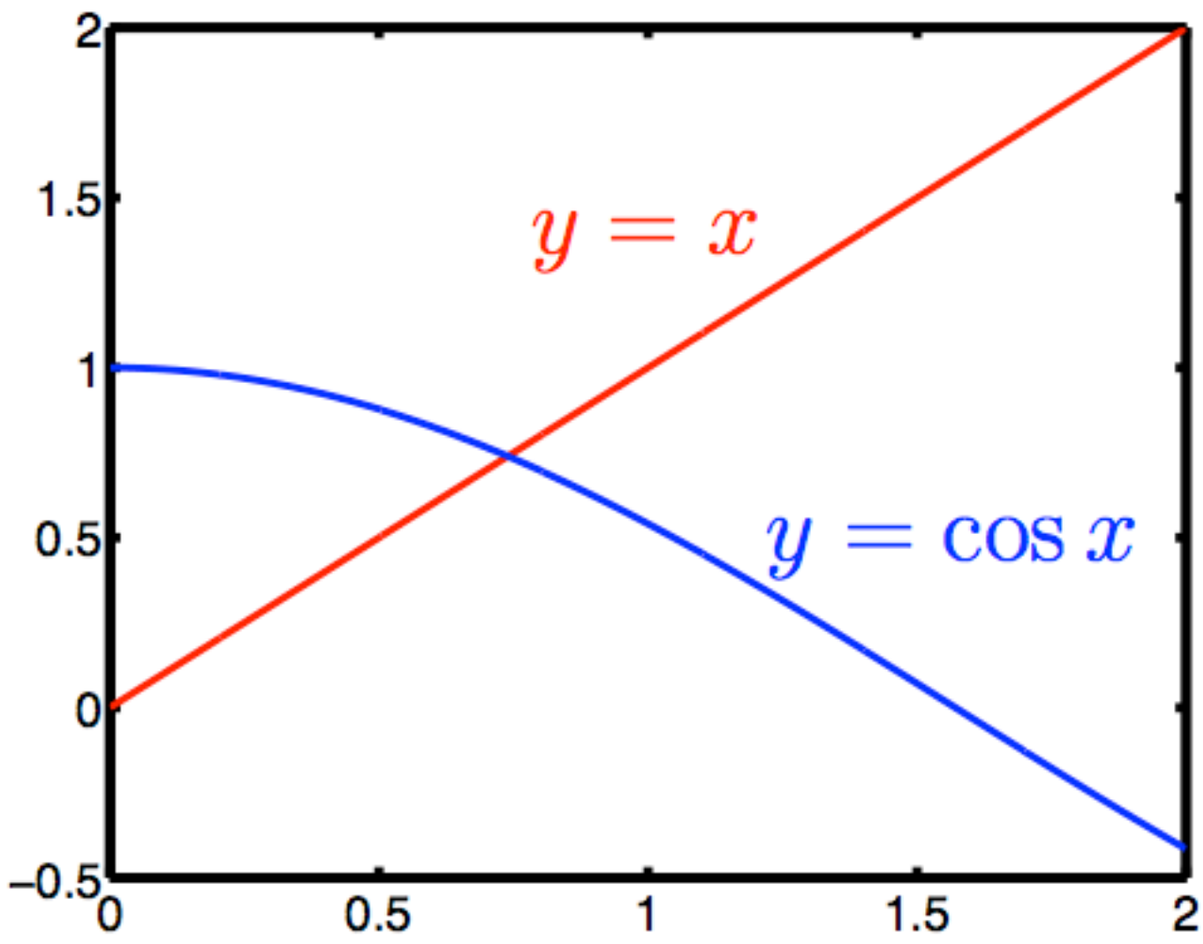
- **Bisection Method:**
- **Newton (-Raphson) Method:**
- **Fix-Point iteration Method:**
- **Secant Method:**

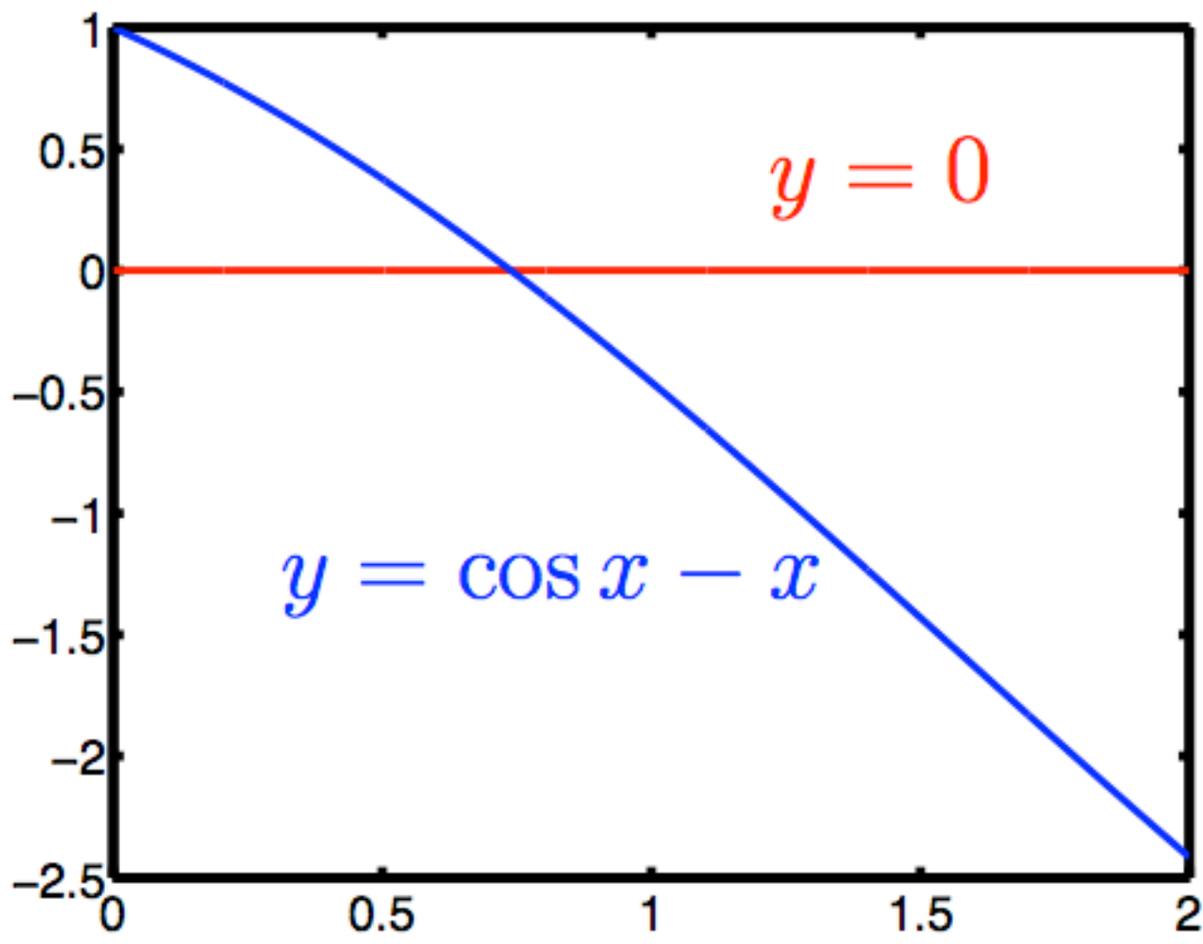
Roots of equation:

$$F(x) = 0$$

Example:

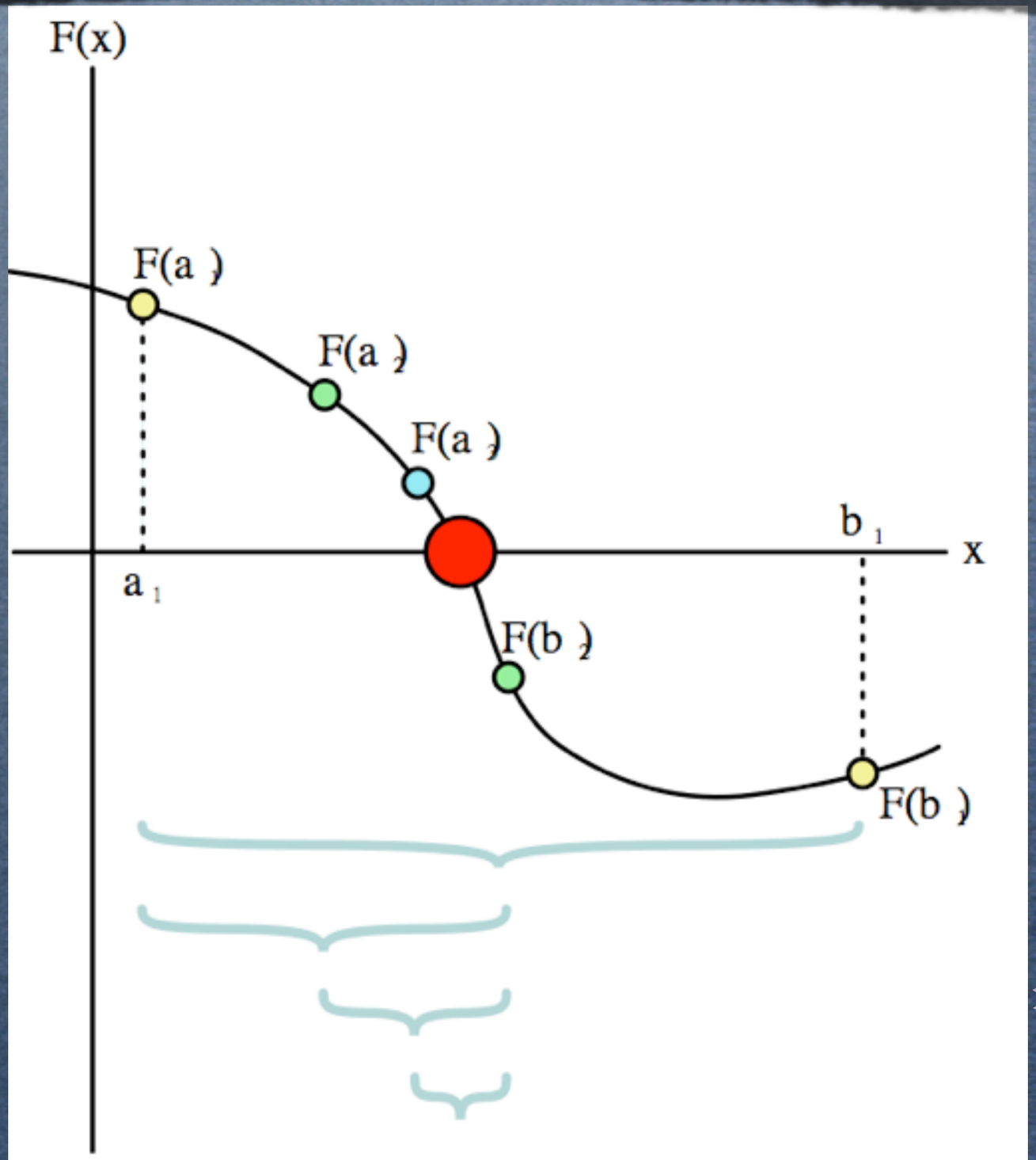
$$F(x) = cos\, x - x = 0$$





Roots of equation: Bisection method

$$F(a) \cdot F(b) < 0$$



Taylor Series: with Cauchy's integral formula

- The Taylor series of a function $f(z)$ is,

$$f(z) = \sum_{n=0}^{\infty} a_n (z - z_0)^n, \qquad \text{where} \qquad a_n = \frac{1}{n!} f^{(n)}(z_0),$$

- By Cauchy's integral formula,

$$a_n = \frac{1}{2\pi\, i} \oint_C \frac{f(z)}{(z - z_0)^{n+1}} \mathrm{d}\, z.$$

Taylor Series: Proof

- The fundamental theorem of calculus states that

$$\int_a^x f'(t)\,\mathrm{d}t = f(x) - f(a),$$

which can be rearranged to,

$$f(x) = f(a) + \int_a^x f'(t)\,\mathrm{d}t.$$

- By integration by parts,

$$\begin{aligned} f(x) &= f(a) + (x-a)f'(a) + \int_a^x (x-t)f''(t)\,\mathrm{d}t, \\ &= f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + \frac{1}{2}\int_a^x (x-t)^2 f'''(t)\,\mathrm{d}t, \end{aligned}$$

- The Taylor series of a function $f(z)$ is,

$$f(z) = \sum_{n=0}^{\infty} a_n (z - z_0)^n, \qquad \text{where} \qquad a_n = \frac{1}{n!} f^{(n)}(z_0),$$

Power Series: Common-used Maclaurin Series

$$\frac{1}{1-x} = 1+x+x^2+\cdots=\sum_{m=0}^{\infty} x^m, \qquad |x|<1$$

$$\frac{1}{1+x} = 1-x+x^2-\cdots=\sum_{m=0}^{\infty} (-1)^m x^m \qquad |x|<1$$

$$e^x = 1+x+\frac{x^2}{2!}+\cdots=\sum_{m=0}^{\infty}\frac{x^m}{m!}, \qquad |x|<\infty.$$

$$\cos x = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\cdots=\sum_{m=0}^{\infty}\frac{(-1)^m x^{2m}}{(2m)!}, \qquad |x|<\infty.$$

$$\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\cdots=\sum_{m=0}^{\infty}\frac{(-1)^m x^{2m+1}}{(2m+1)!}, \qquad |x|<\infty.$$

$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\cdots=\sum_{m=1}^{\infty}\frac{(-1)^{m-1} x^m}{m}, \qquad -1<x\le 1.$$

Complex numbers:

x -> z

Roots of equation: Newton's method

Taylor expansion to the 1st-order:

$$f(x) = f(x_0) + \left.\frac{df(x)}{dx}\right|_{x=x_0} (x - x_0) + \mathbf{O}(\Delta x^2)$$

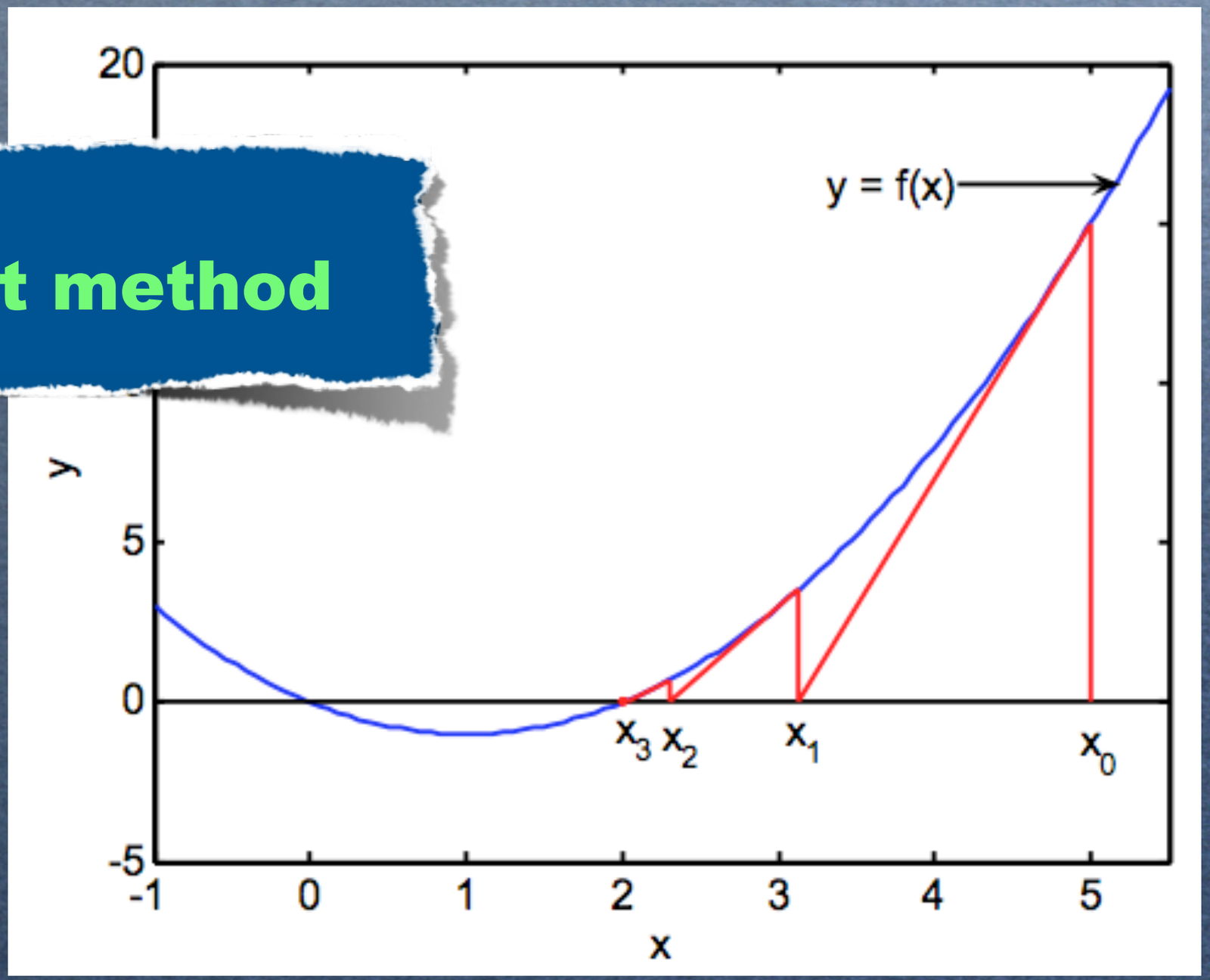
- Newtwon's method:

$$\begin{aligned} x_{\text{new}} &= x_{\text{old}} + \Delta x, \\ &= x_{\text{old}} - \frac{f(x_0)}{f'(x_0)} \end{aligned}$$

Roots of equation: Newton's method

Example: $f(x) = x^2 - 2x = 0, \quad \text{with} \quad f(x_0) = 5$

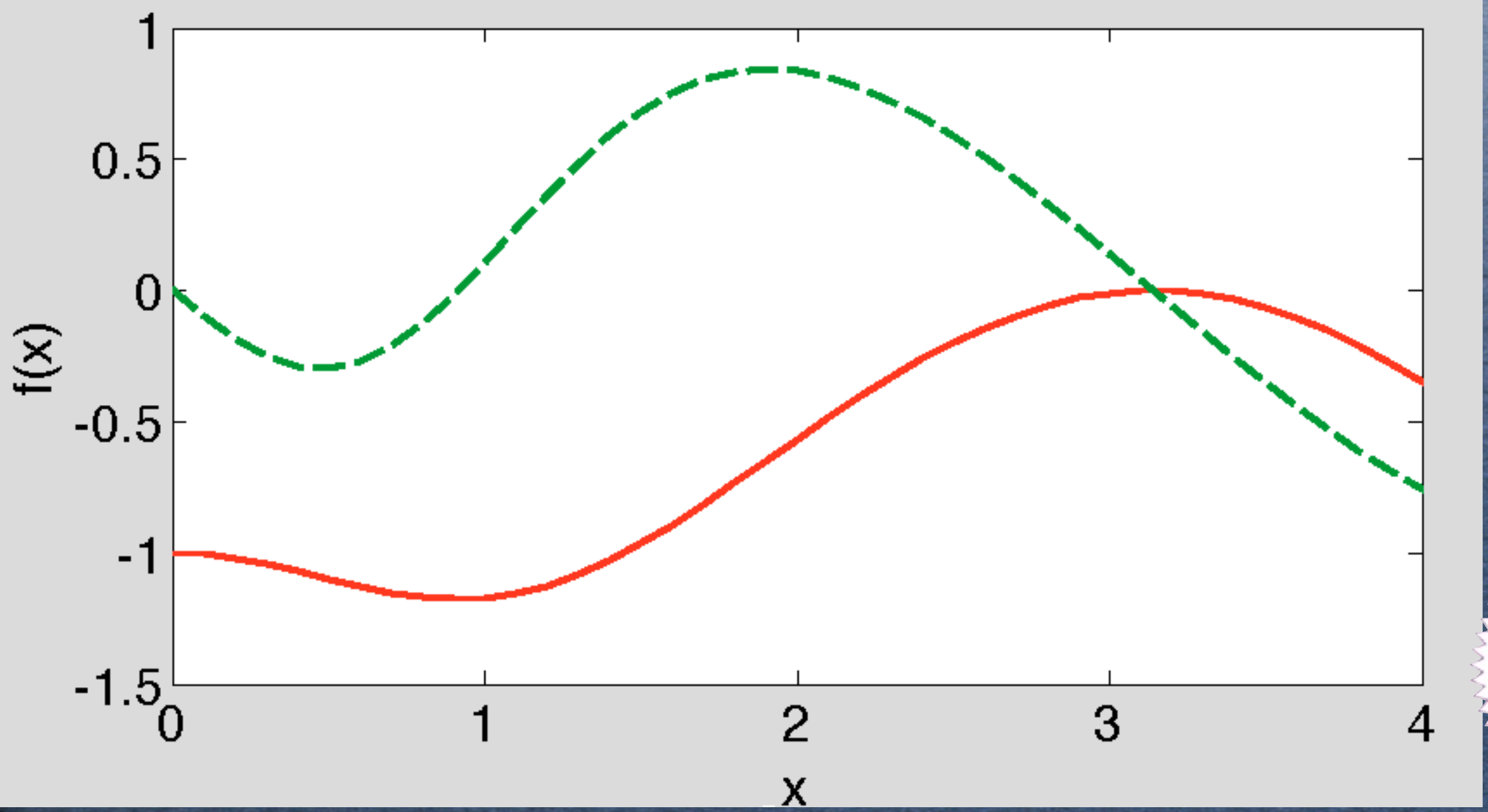
tangent method



Homework 1: Newton's method

Example:

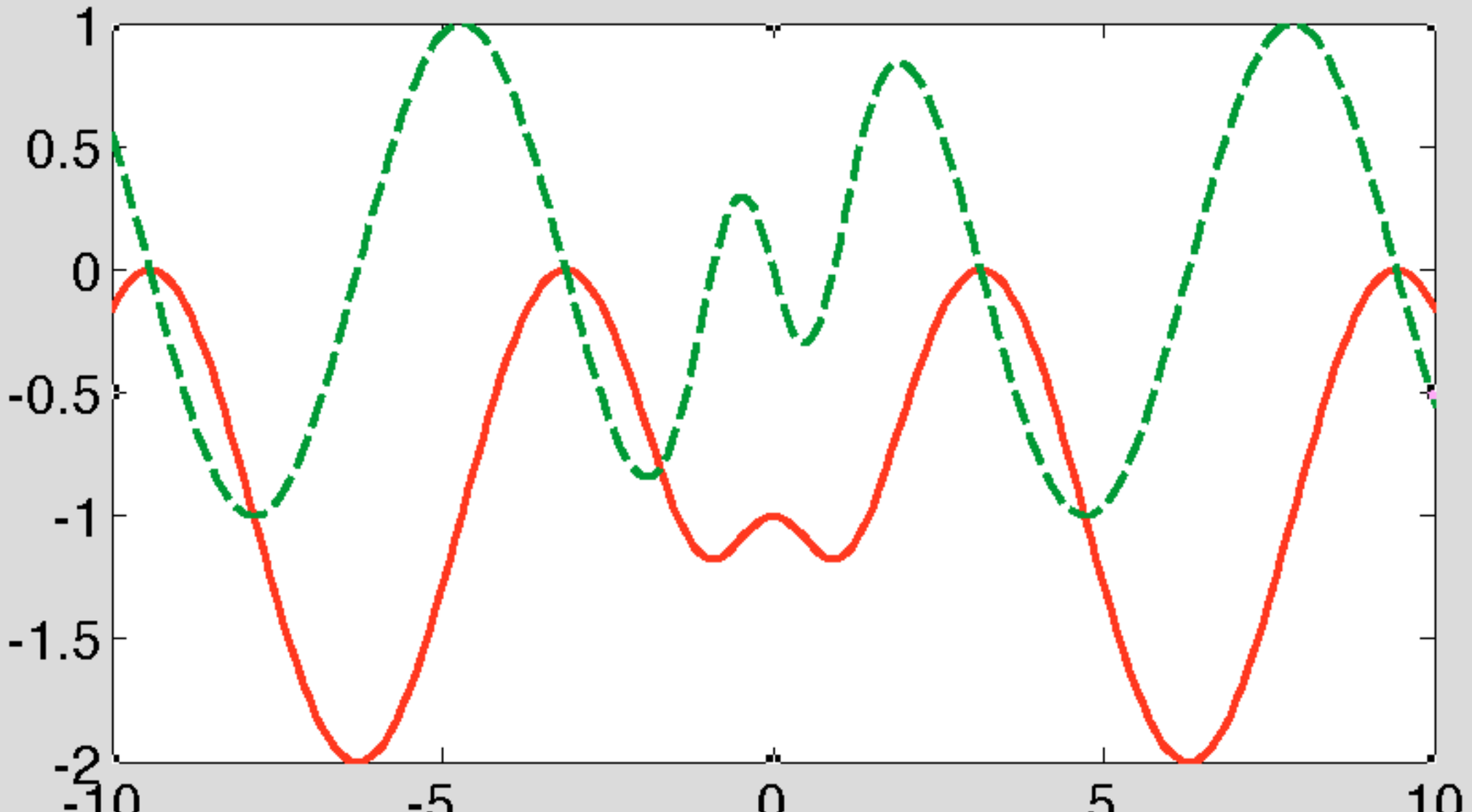
$$e^{-x^2} = cos\, x + 1$$



Homework 1: Newton's method

Example:

$$e^{-x^2} = cos\, x + 1$$



More than 1D: Newton's method

Example:

$$\begin{aligned} f_1 &= x_1^3 + x_2^2 = 0, \\ f_2 &= x_1^2 - x_2^3 = 0. \end{aligned}$$

- define the vector $\vec{F}$:

$$\vec{F} \equiv \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} x_1^3 + x_2^2 \\ x_1^2 - x_2^3 \end{bmatrix}.$$

More than 1D: Newton's method

$$\vec{F}(x_{i+1}) = \begin{bmatrix} f_1(x_i) \\ f_2(x_i) \\ \vdots \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}_{x=x_i} \begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \end{bmatrix}.$$

$$= \vec{F}(x_i) + \bar{\bar{J}} \cdot d\vec{x}$$

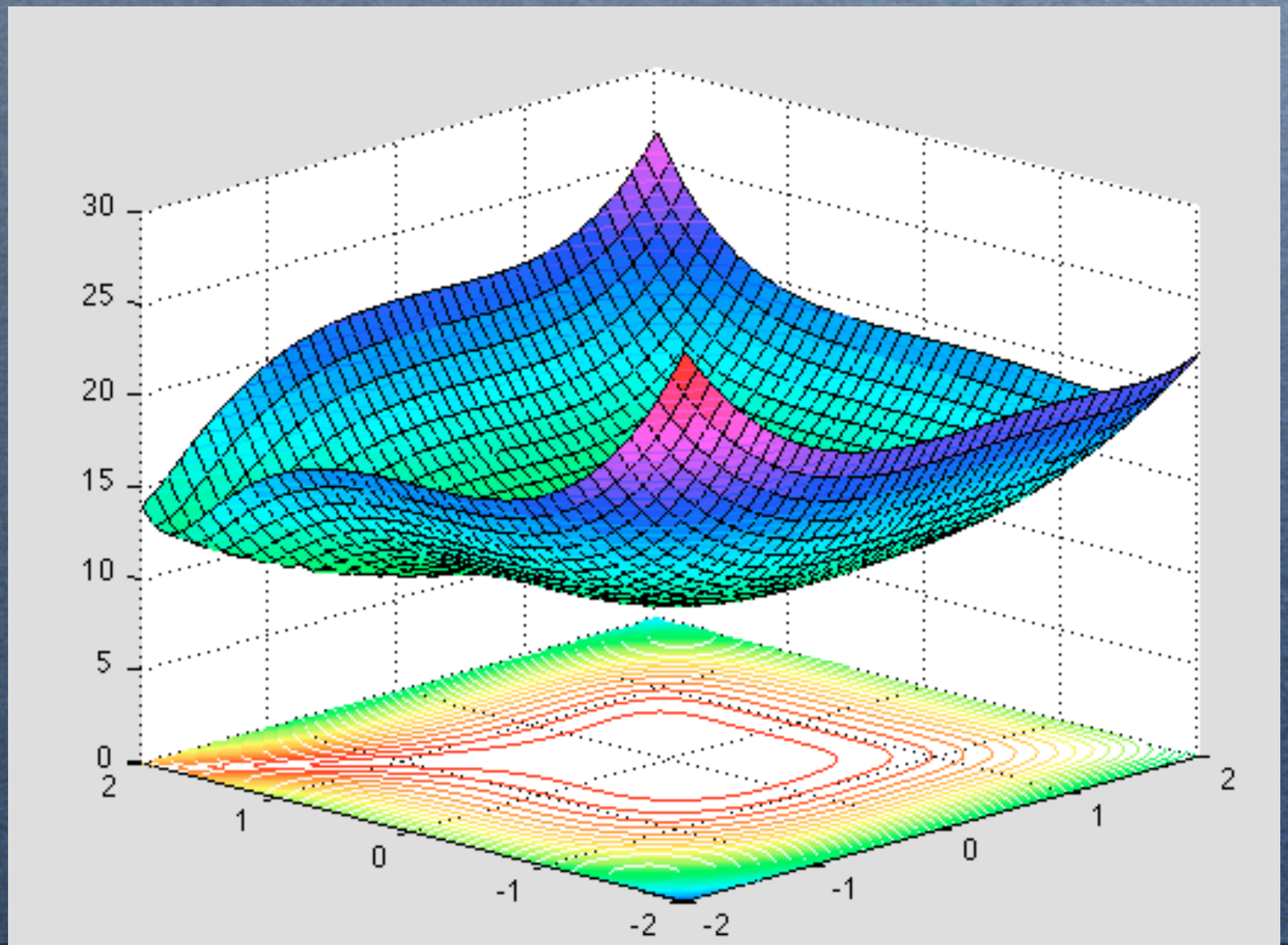
where $\bar{\bar{J}}$ is called the *Jacobian matrix*.

$$\vec{X}_{i+1} = \begin{bmatrix} x_1^{i+1} \\ x_2^{i+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1^{i} \\ x_2^{i} \\ \vdots \end{bmatrix} - \frac{\vec{F}(x_i)}{\bar{\bar{J}}(x_i)}$$

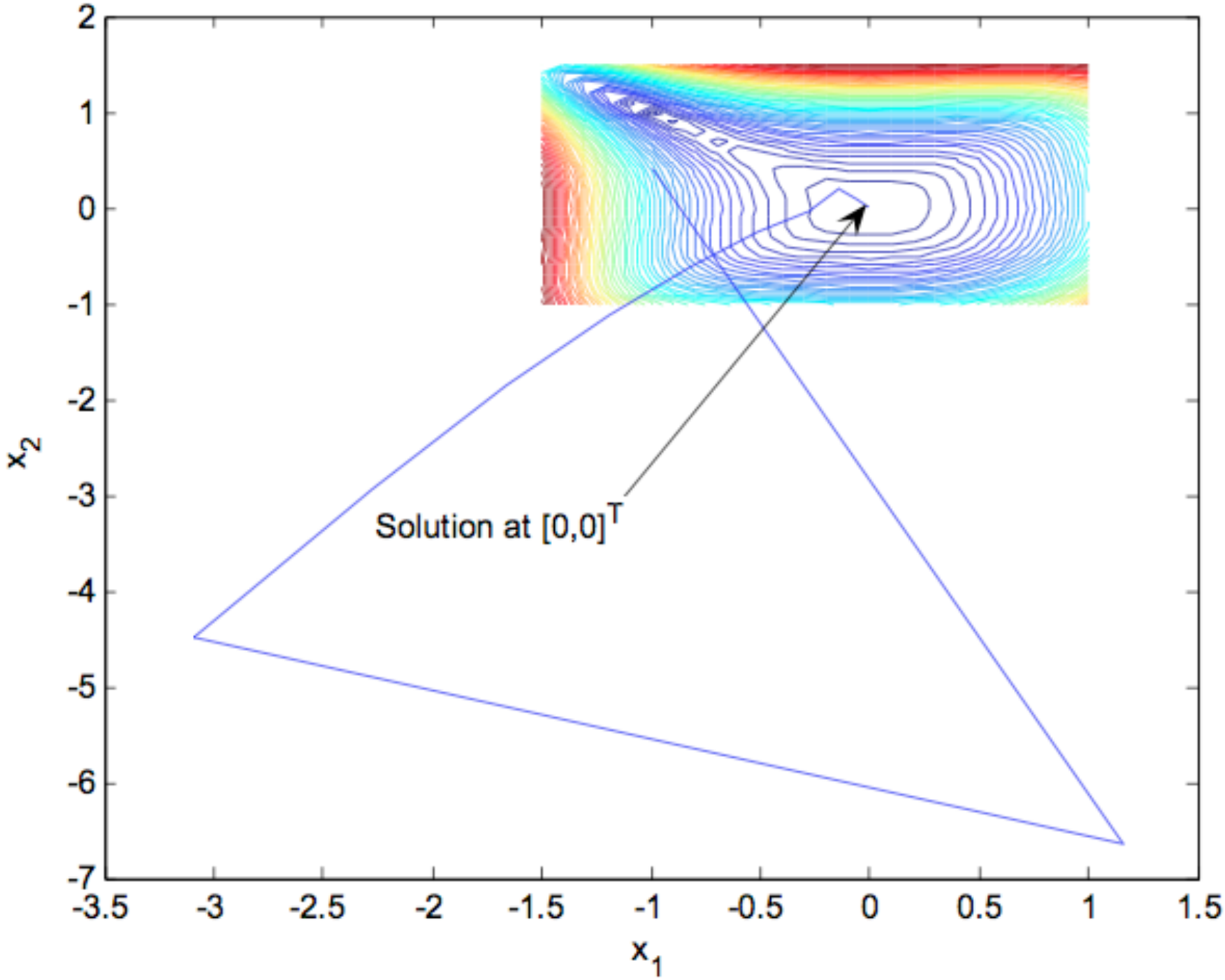
More than 1D: Newton's method

Example:

$$\begin{aligned} f_1 &= x_1^3 + x_2^2 = 0, \\ f_2 &= x_1^2 - x_2^3 = 0. \end{aligned}$$



More than 1D: Newton's method



Homework 1: Newtown's method

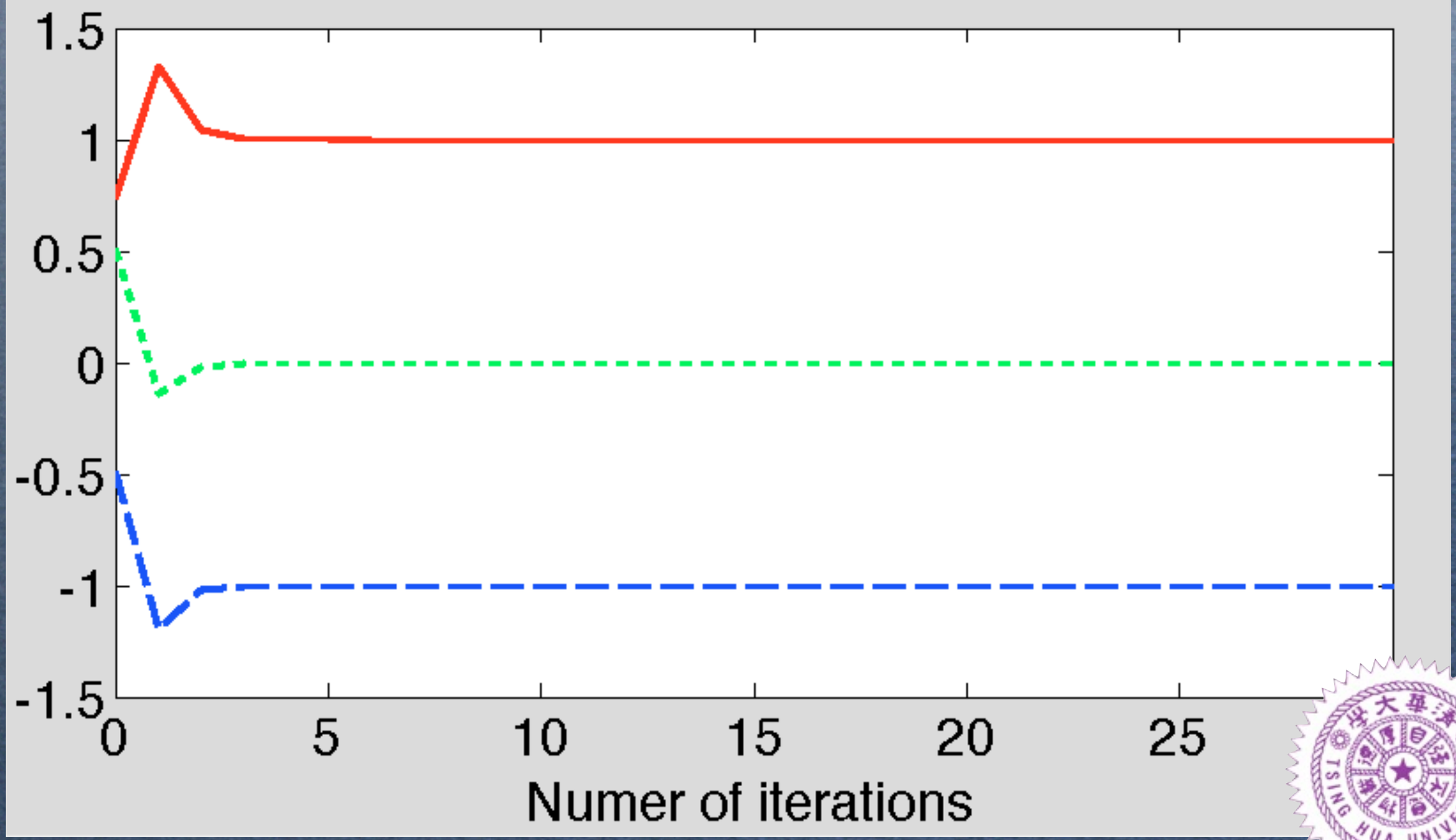
Example:

$$\begin{cases} x+y+z=0 \\ x^2+y^2+z^2=2 \\ x(y+z)=-1 \end{cases},$$

$$\vec{F}(\vec{X}) \equiv \begin{bmatrix} x+y+z \\ x^2+y^2+z^2-2 \\ x(y+z)+1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

$$\vec{X}_{n+1} = \begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} - \frac{\vec{F}(\vec{X}_n)}{\vec{\vec{J}}(\vec{X}_n)}$$

Homework 1: Newtown's method



Newton's method: Jacobian matrix

- The Jacobian of the inverse transformation is the reciprocal of the Jacobian of the original transformation, *i.e.*,

$$\left|\frac{\partial(x,y)}{\partial(u,v)}\right| = \frac{1}{\left|\frac{\partial(u,v)}{\partial(x,y)}\right|}$$

- This is a consequence of the fact that the determinant of the inverse of a matrix $\bar{\bar{A}}$ is the reciprocal of the determinant of $\bar{\bar{A}}$.

Jacobian matrix: Singular Value Decomposition

Singular Jacobian matrix

- The singular value decomposition, SVD, of an $m \times n$ real or complex matrix $\bar{\bar{M}}$ is a factorization of the form

$$\bar{\bar{M}} = \bar{\bar{U}}\bar{\bar{\Sigma}}\bar{\bar{V}}^*$$

where $\bar{\bar{U}}$ is an $m \times m$ real or complex unitary matrix, $\bar{\bar{\Sigma}}$ is an $m \times n$ rectangular diagonal matrix with nonnegative real numbers on the diagonal, and $\bar{\bar{V}}^*$ (the conjugate transpose of $\bar{\bar{V}}$) is an $n \times n$ real or complex unitary matrix.

- The diagonal entries $\bar{\bar{\Sigma}}_i$, i of $\bar{\bar{\Sigma}}$ are known as the singular values of $\bar{\bar{M}}$.

Homework 1: Newtown's method

- Define $z = -(x + y)$,
- then

$$\begin{cases} x^2 + y^2 + (x+y)^2 = 2 \\ x(y - x - y) = -1 \end{cases},$$

Homework 1: Newtown's method

$$
\begin{aligned}
x^2 + y^2 + z^2 &= 2 \\
\therefore x &= \sqrt{2}\sin\theta\cos\phi \\
y &= \sqrt{2}\sin\theta\sin\phi \\
z &= \sqrt{2}\cos\theta
\end{aligned}
\tag{7}
$$

that means we eliminate one parameter and change coordinate into spherical one by using θ and ϕ two parameters. Those are limited by $\theta = [0, \pi]$ and $\phi = [0, 2\pi]$.
Define

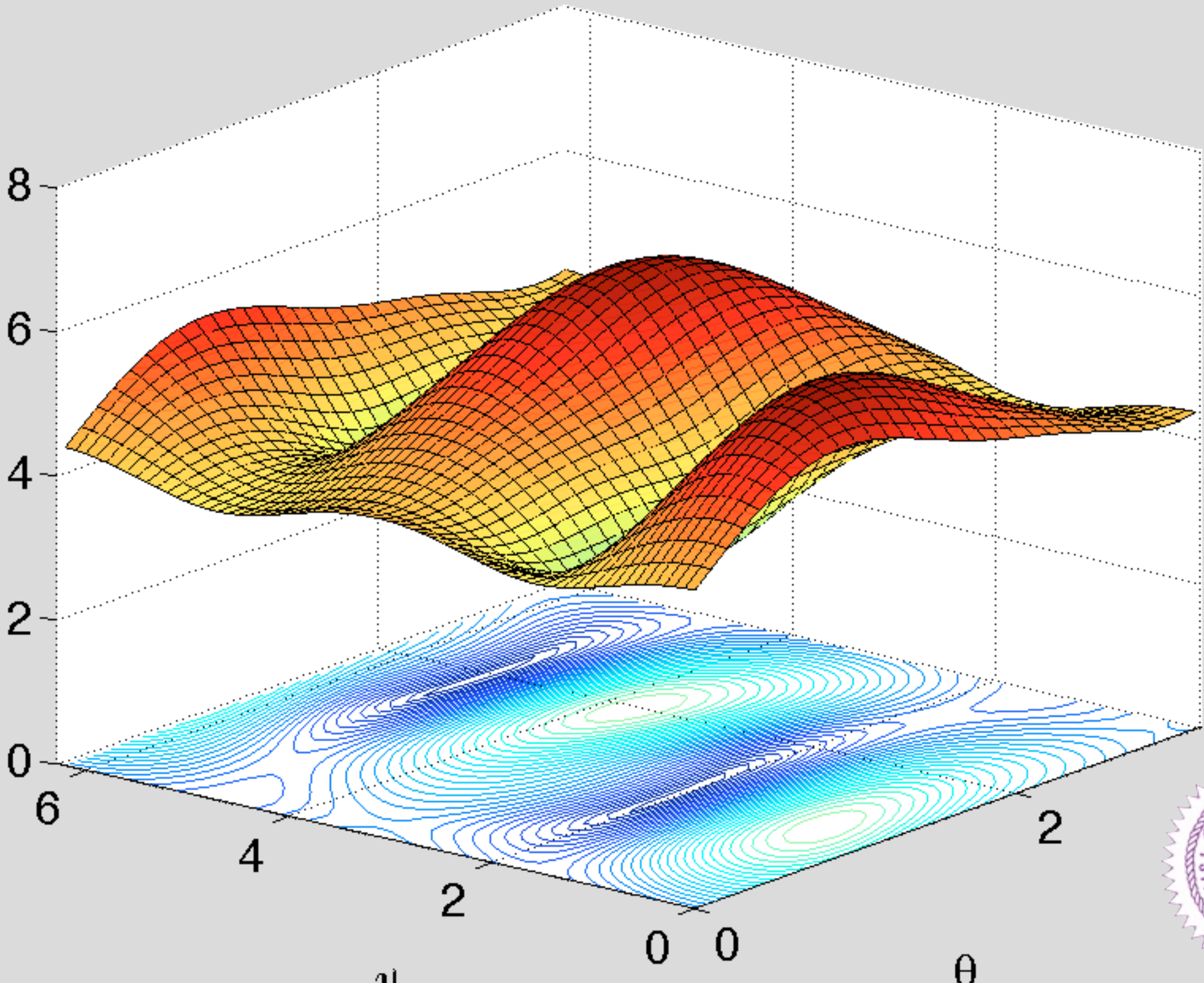
$$
\begin{aligned}
f(\theta, \phi) &= \sin\theta\cos\phi + \sin\theta\sin\phi + \cos\phi \\
g(\theta, \phi) &= 1 + 2\sin\theta\cos\phi(\sin\theta\sin\phi + \cos\theta)
\end{aligned}
\tag{8}
$$

Newton's method for solving two nonlinear equations would be

$$
\begin{aligned}
\theta_{n+1} &= \theta_n - \frac{f g_\phi - g f_\phi}{f_\theta g_\phi - g_\theta f_\phi} \\
\phi_{n+1} &= \phi_n - \frac{f_\theta g - g_\theta f}{f_\theta g_\phi - g_\theta f_\phi}
\end{aligned}
\tag{9}
$$

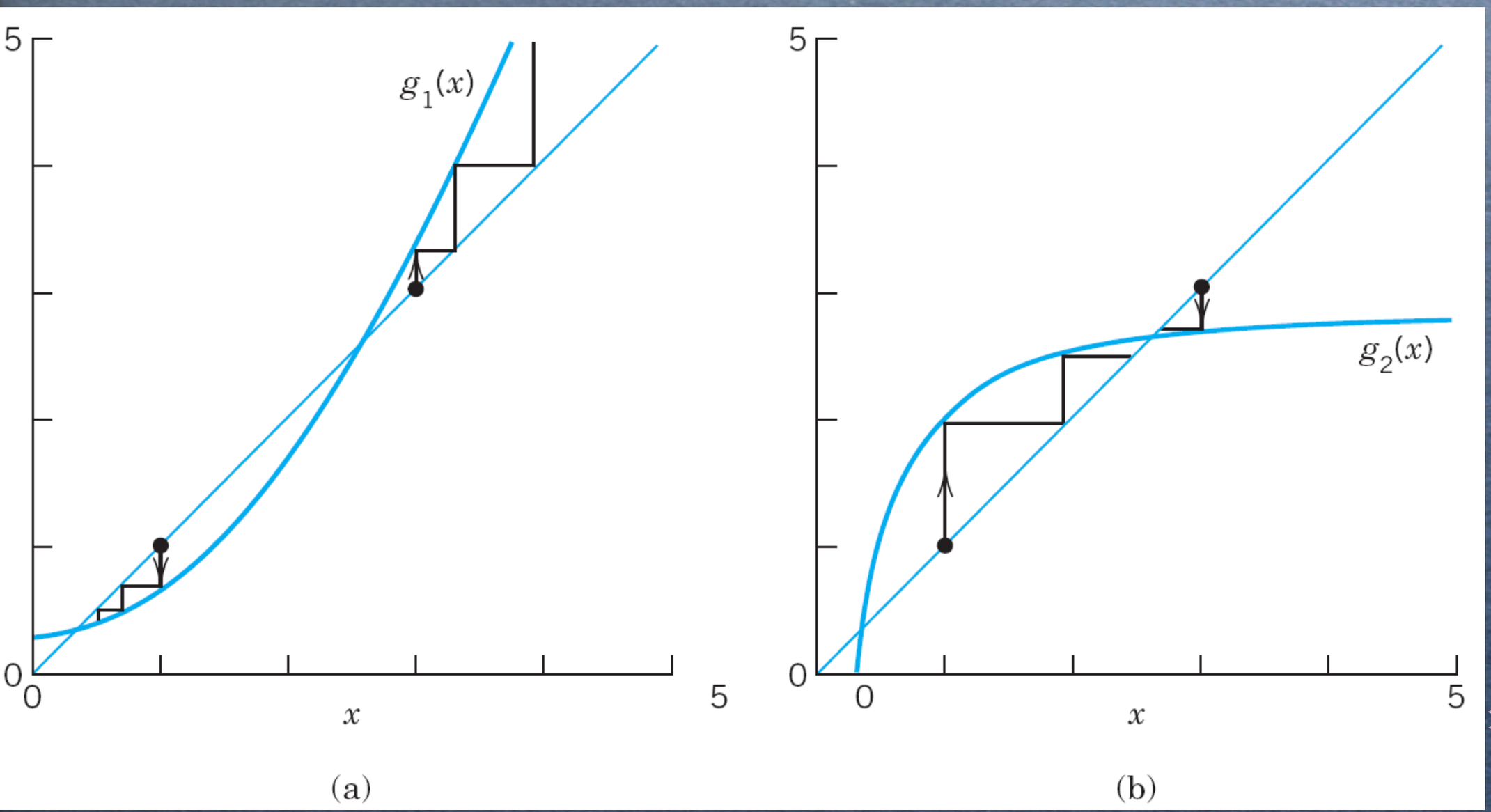
Right now, we face a new problem: the starting values! But since our coordinate θ and ϕ are limited, our search can be done by point by point, that means we divide small points within $\theta = [0, \pi]$ and $\phi = [0, 2\pi]$. Conclusion: there are total four solutions,

$$
\begin{aligned}
(x, y, z) &= (-1, 0, 1) \\
(x, y, z) &= (-1, 1, 0) \\
(x, y, z) &= (1, 0, -1) \\
(x, y, z) &= (1, -1, 0)
\end{aligned}
\tag{10}
$$


8
6
4
2
0
6
4
2
0
0
2
ψ
θ

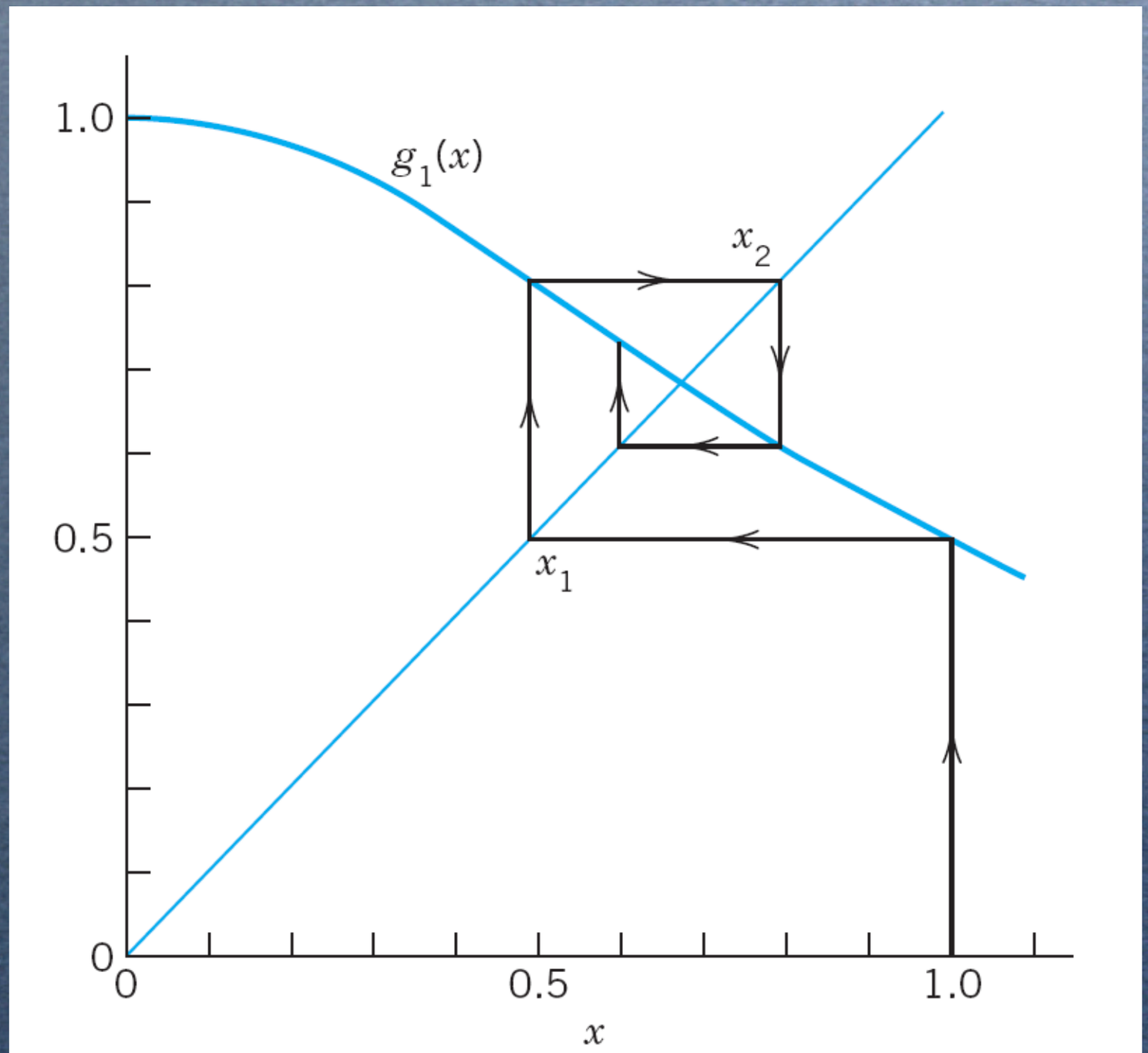
Roots of equation: Fixed-point method

Example: $F(x) = 0 \qquad \Rightarrow \qquad x_{n+1} = g(x_n)$



Roots of equation: Fixed-point method

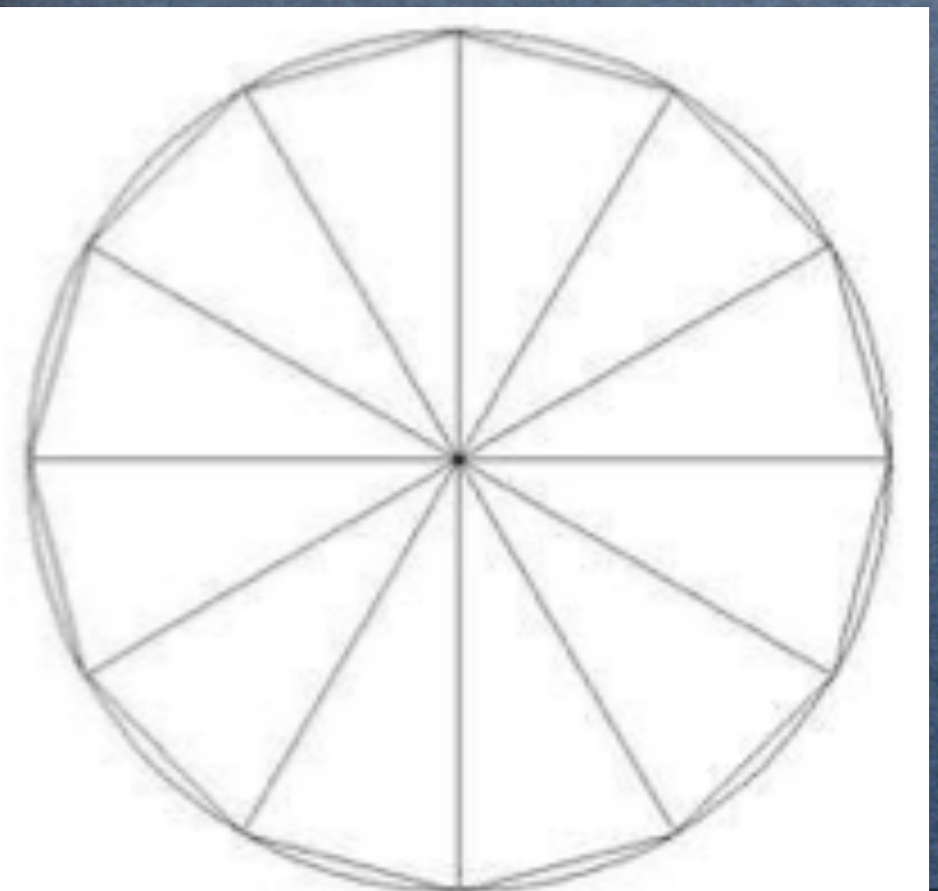
Example:



Homework 1: Find the value of π

- The length of the curved part of a unit circle is 2π.
- We can approximate π by using triangles. Consider the arc bisected as shown in the figure. 1.
- The length of the hypotenuse of the triangle is $2\sin(\theta/2)$ for a fraction of the circle $1/2\,k$, then an approximation for π is

$$\pi \approx 2\,k\,\sin(\theta/2).$$



Homework 1: Find the value of π

- From the trigonometry, we have

$$\sin^2\frac{1}{2}\theta = \frac{1}{2}(1-\cos\theta) = \frac{1}{2}(1-\sqrt{1-\sin^2\theta}) = \frac{\sin^2\theta}{2+2\sqrt{1-\sin^2\theta}}.$$

- Now let θ_n be the angle that results from division of the circular arc into 2^n pieces.

- Next let $S_n = \sin^2\theta_n$, and assign that

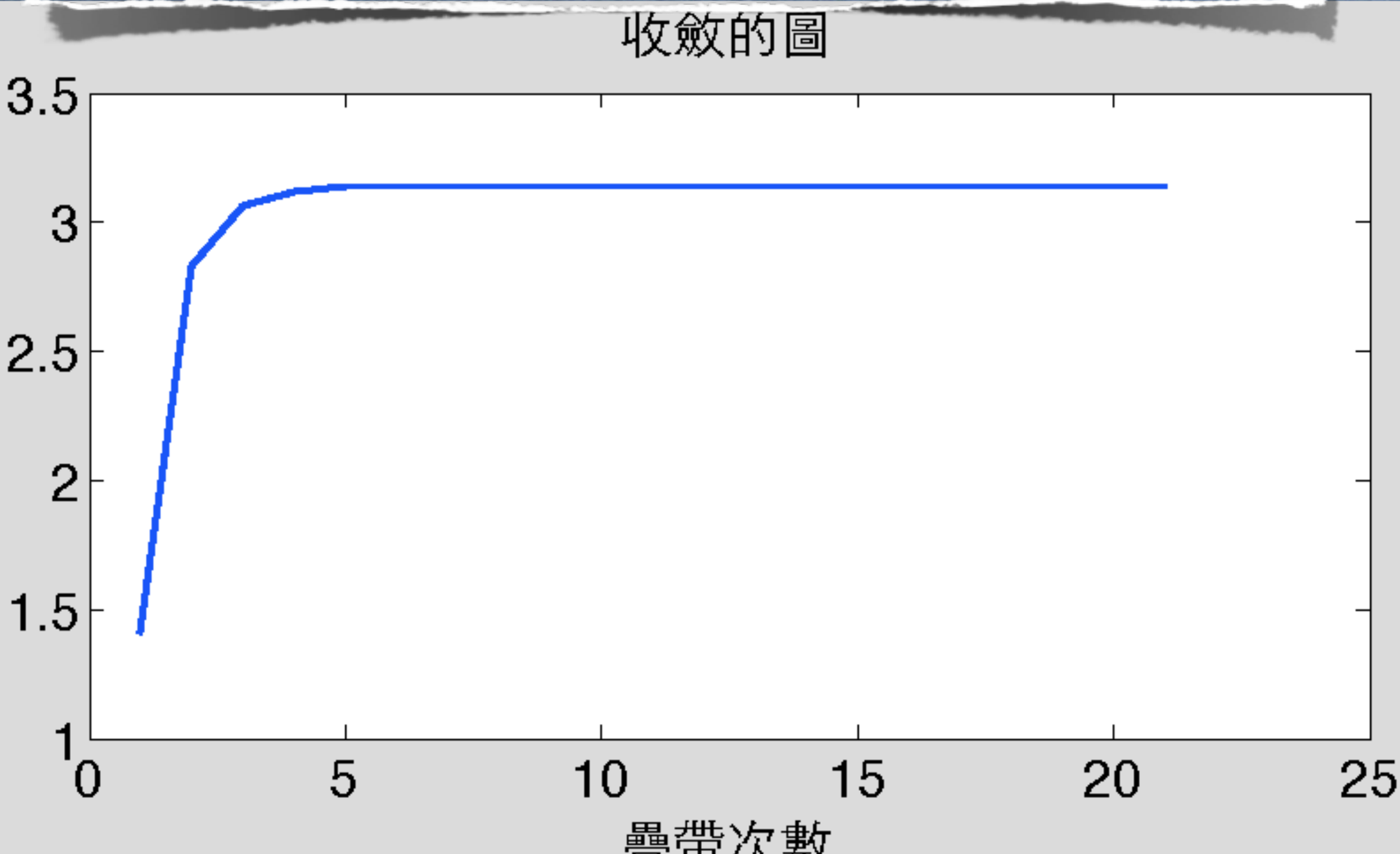
$$\boxed{S_{n+1} = S_n/(2+2\sqrt{1-S_n}).}$$

- Then π can be approximated by

$$\boxed{\pi \approx P_n \equiv 2^n\sqrt{S_{n+1}}.}$$

- Starting with $S_2 = 1$ and $P_1 = 2$, compute S_{n+1} and P_n recursively for $2 \le n \le 20$.

Homework 1: Find the value of π



Homework 1: Find the value of π

```
2-iterations, with the error: 0.313166
3-iterations, with the error: 0.0801252
4-iterations, with the error: 0.0201475
5-iterations, with the error: 0.00504416
6-iterations, with the error: 0.0012615
7-iterations, with the error: 0.000315403
8-iterations, with the error: 7.88524e-05
9-iterations, with the error: 1.97132e-05
10-iterations, with the error: 4.92831e-06
11-iterations, with the error: 1.23208e-06
12-iterations, with the error: 3.0802e-07
13-iterations, with the error: 7.70049e-08
14-iterations, with the error: 1.92512e-08
15-iterations, with the error: 4.81281e-09
16-iterations, with the error: 1.2032e-09
17-iterations, with the error: 3.008e-10
18-iterations, with the error: 7.52003e-11
19-iterations, with the error: 1.88001e-11
20-iterations, with the error: 4.6998e-12
21-iterations, with the error: 1.17506e-12
22-iterations, with the error: 2.93543e-13
23-iterations, with the error: 7.28306e-14
24-iterations, with the error: 1.77636e-14
25-iterations, with the error: 3.9968e-15
26-iterations, with the error: 4.44089e-16
27-iterations, with the error: -4.44089e-16
28-iterations, with the error: -4.44089e-16
29-iterations, with the error: -4.44089e-16
30-iterations, with the error: -4.44089e-16
```

Matlab®: eps

Spacing of floating point numbers,

```
>> eps

ans =

    2.2204e-16
```

Matlab®: some tips

- use *matrix/vector* operations rather than *loop* operations
- use *build-in* routine as often as possible
- build your self sub-routines, *.m* files
- use adaptive input argument list
- use the *breakpoint* in the debug mode
- check *workspace* for the type/value of the variables
- remark the code as much as possible

Week 2: (3/13)

2. Root of equations: [T] Ch. 19.2; [C] Ch. 3

- Bisection method,
- Fixed-point iteration method,
- Newton's method,
- Newton's method for nonlinear systems.
- Secant method.

3. Interpolation: [T] Ch. 19.3; [C] Ch. 4

- Lagrange polynomial
- Runge phenomena
- Chebyshev interpolation
- Newton's divided difference interpolation
- spline interpolation
- Padé interpolation

4. Numerical Integration: [T] Ch. 19.5; [C] Ch. 5

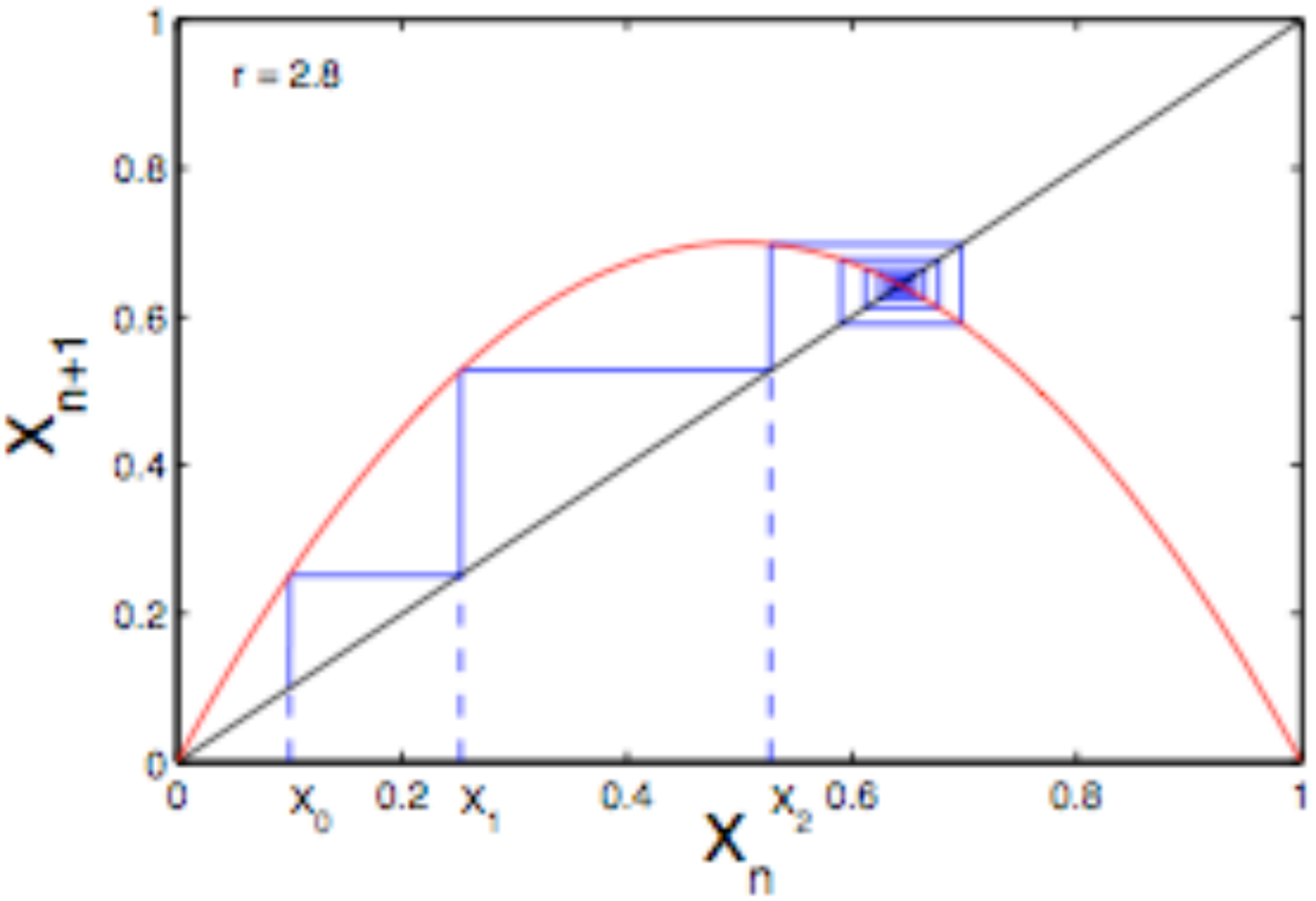
- Trapezoidal rule
- Simpson's rule
- Adaptive method
- Gauss integration formula

Fixed-point method: Logistic equation

$$\frac{dN}{dt} = \gamma N\left(1 - \frac{N}{K}\right)$$

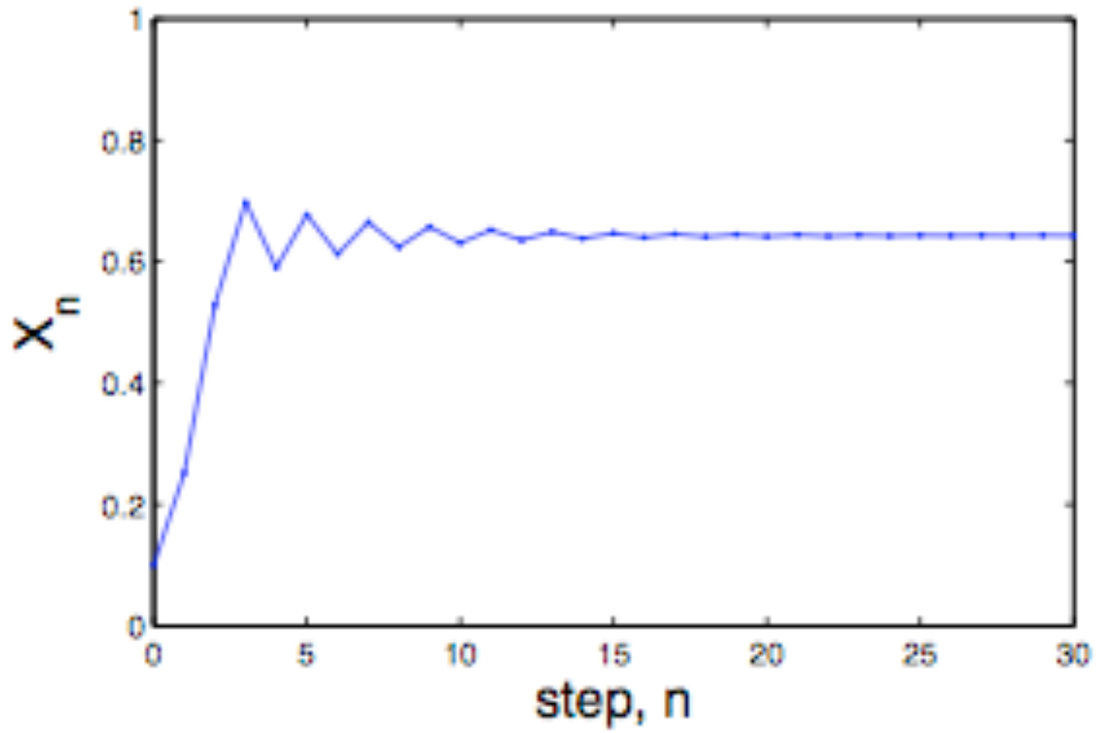
$$\frac{dX}{dt} = \gamma X(1 - X), \qquad X \equiv N/K$$

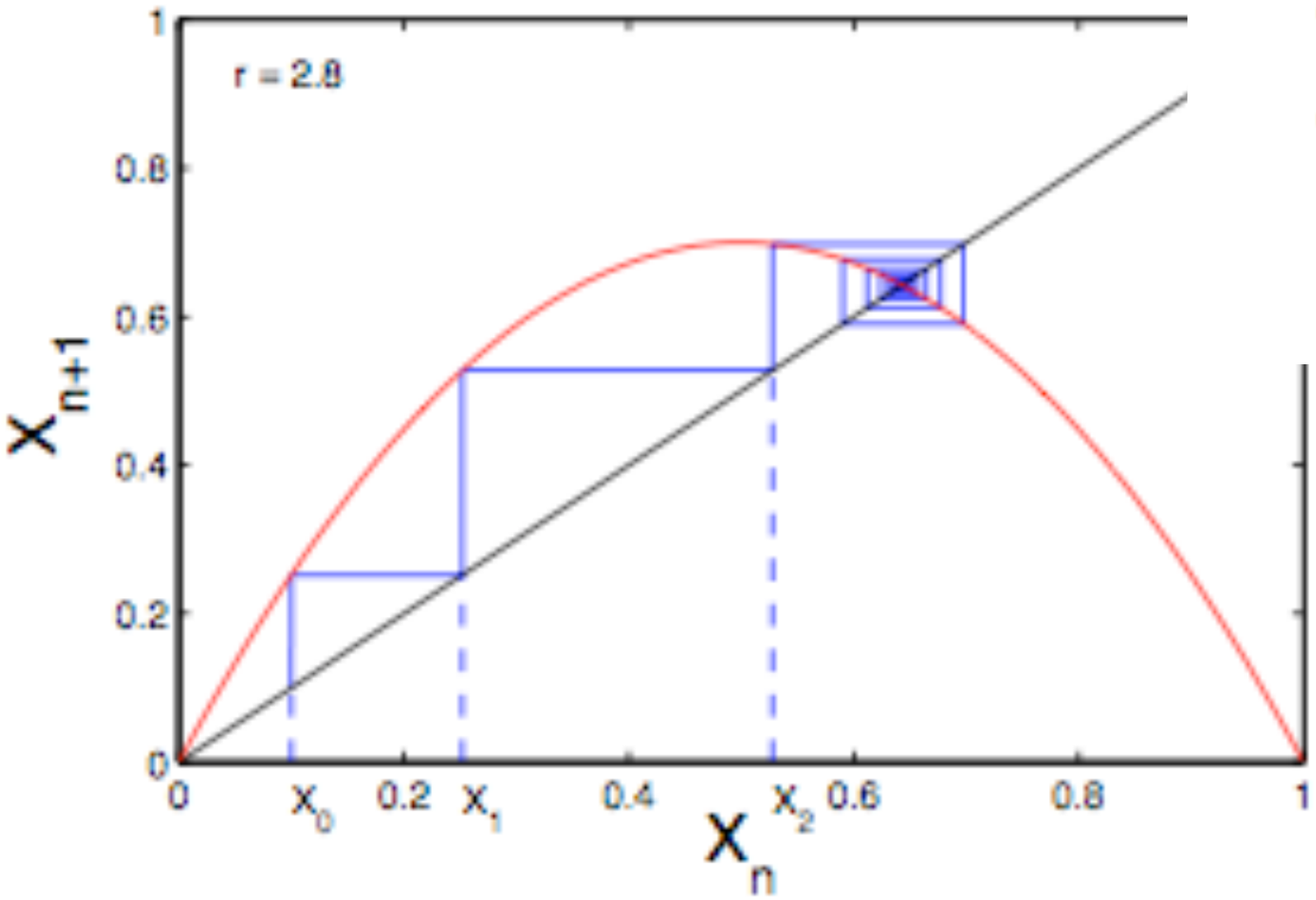
$$X_{n+1} = \gamma X_n (1 - X_n)$$



Fixed-point method: Logistic equation

$$X_{n+1} = \gamma X_n (1 - X_n)$$





Logistic equation: Stabilities

- Steady state (or equilibrium):

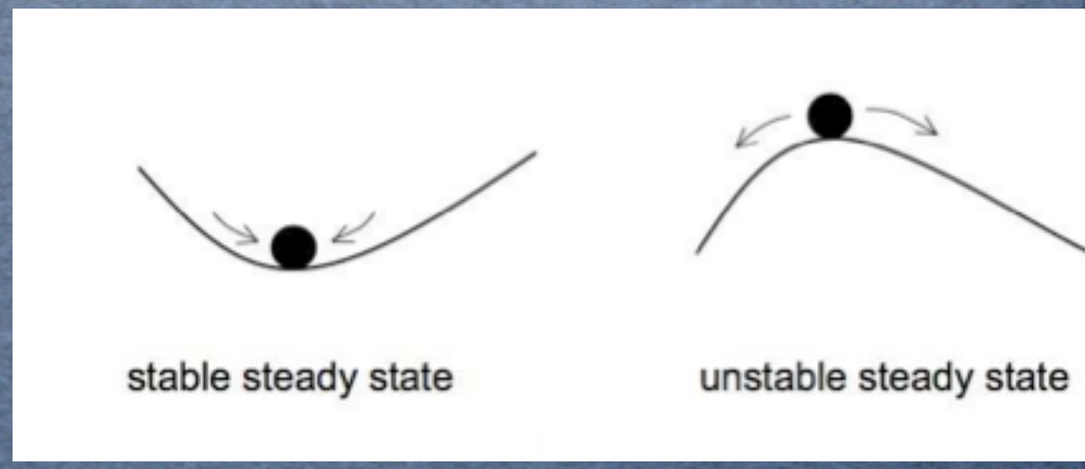
$$X_{n+1} = X_n = X_{ss}$$

- Logistic equation, the steady state is given by

$$rX_{ss}^2 - X_{ss}(r-1) = 0,$$

with two steady states are possible:

$$X_{ss1} = 0 \quad \text{and} \quad X_{ss2} = 1 - 1/r$$



- stable steady state is a state that can be reached from the neighbor states,
- unstable steady state is a state that the system will leave as soon as a small perturbation is driven.

Logistic equation: Stabilities

- Stability is a local property.
- By applying a small perturbation x_n that moves the system out of its steady state,

$$X_{ss} \to X_n = X_{ss} + x_n$$

- then

$$\begin{aligned} x_{n+1} &= X_{n+1} - X_{ss} \\ &= f(X_{ss}) + (\frac{df}{dx})_{X=X_{ss}} x_n + \mathbf{O}(x_n^2) - X_{ss} \\ &\approx (\frac{df}{dx})_{X=X_{ss}} x_n \equiv a\, x_n \end{aligned}$$

- If $|a \equiv (\frac{df}{dx})_{X=X_{ss}}| < 1$, the steady state is stable.
- If $|a \equiv (\frac{df}{dx})_{X=X_{ss}}| > 1$, the steady state is unstable.

Logistic equation: Stabilities

- For the logistic equation:

$$X_{n+1} = \gamma X_n (1 - X_n)$$

- For the steady state X_{ss1},

$$a = (r - 2rX)_{X=0} = r,$$

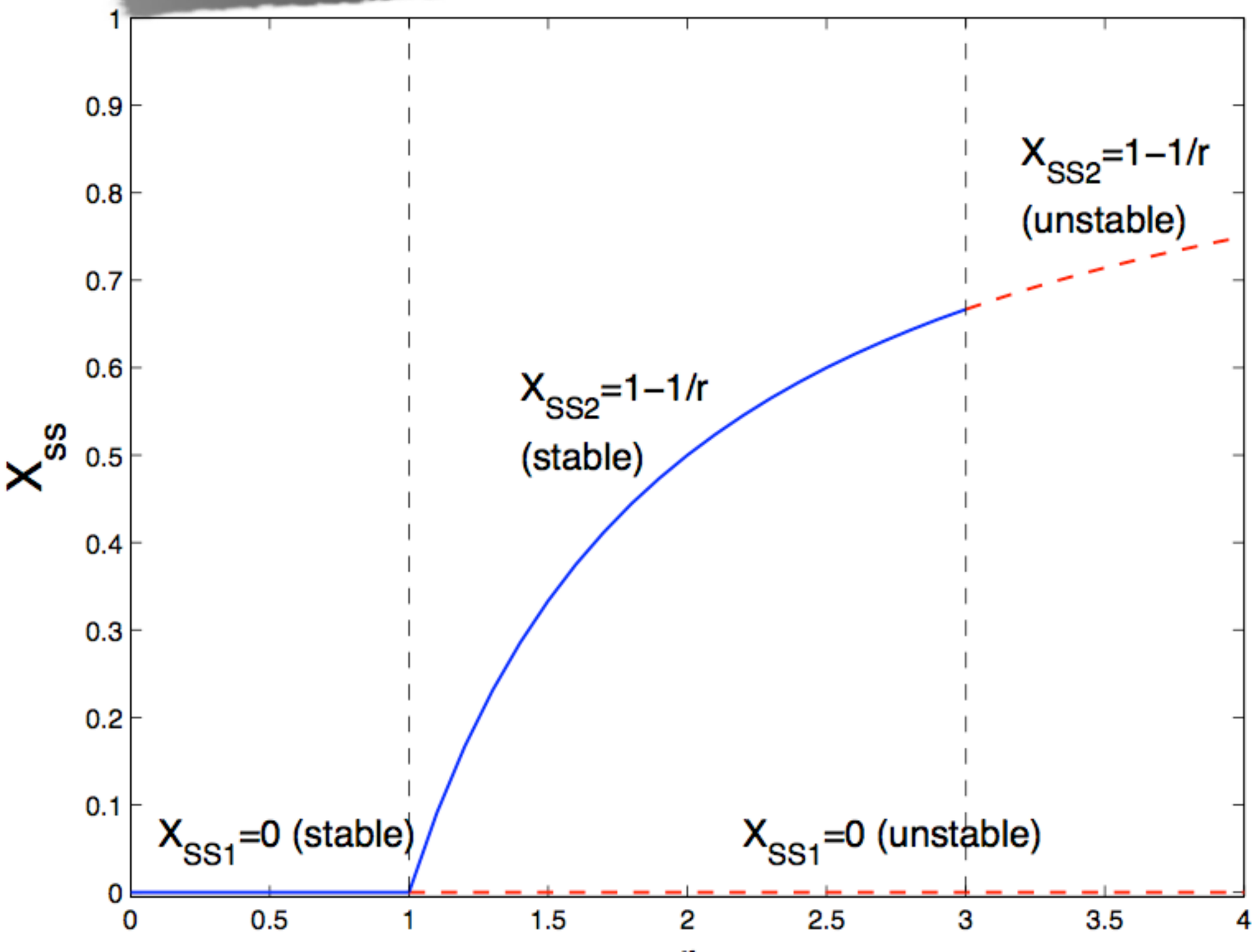
which is stable if $r < 1$.

- For the steady state X_{ss2},

$$a = (r - 2rX)_{X=0} = 2 - r,$$

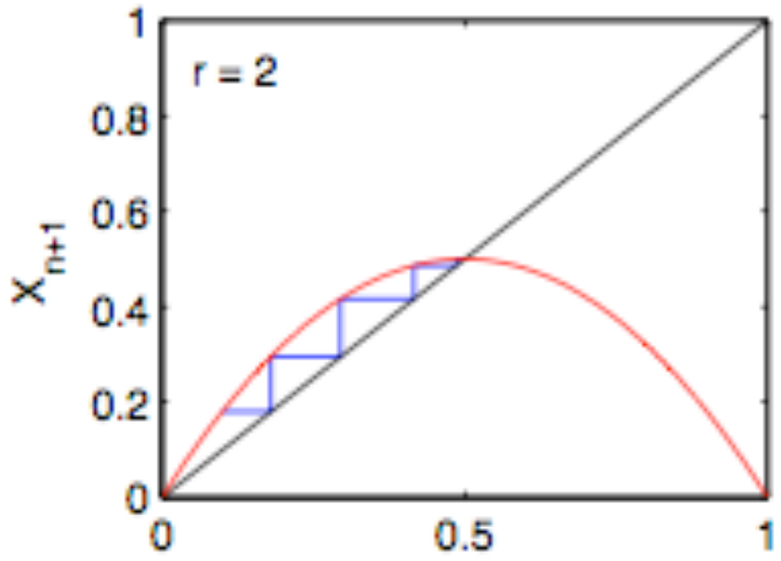
which is stable if $1 < r < 3$.

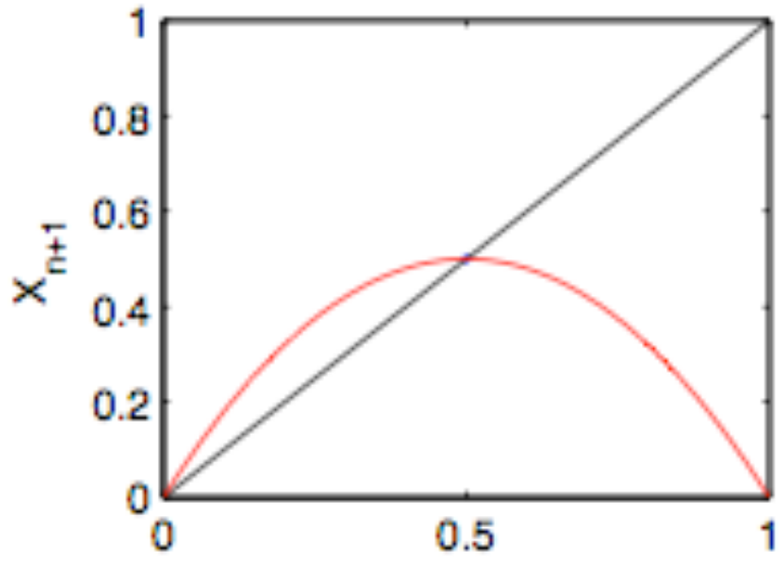
Logistic equation: Stabilities

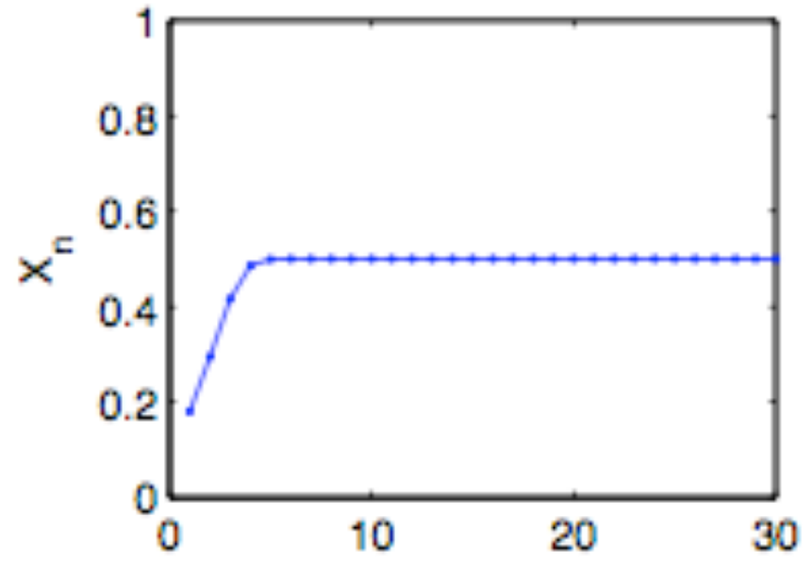


Fixed-point method: Logistic equation

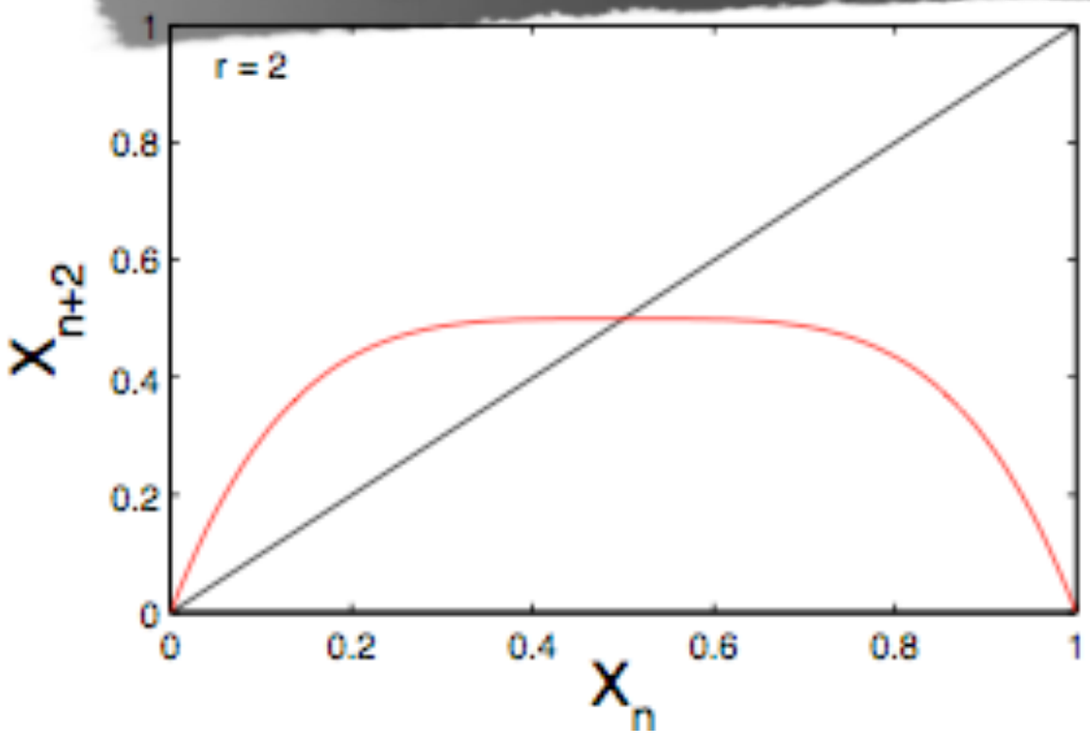
$$X_{n+1} = \gamma\, X_n\,(1 - X_n)$$

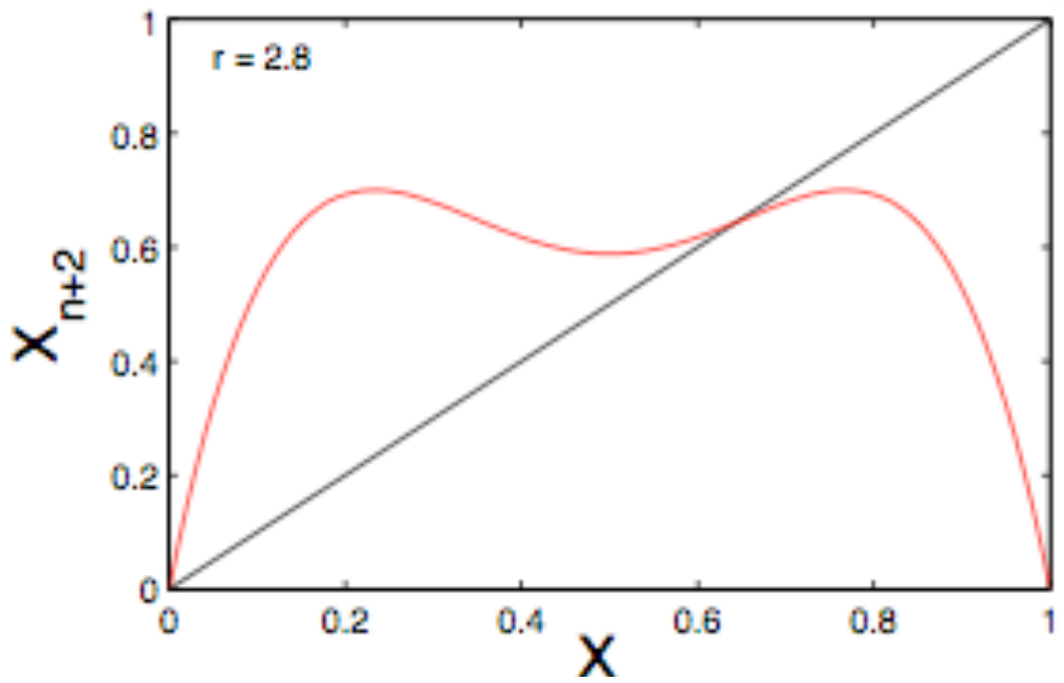


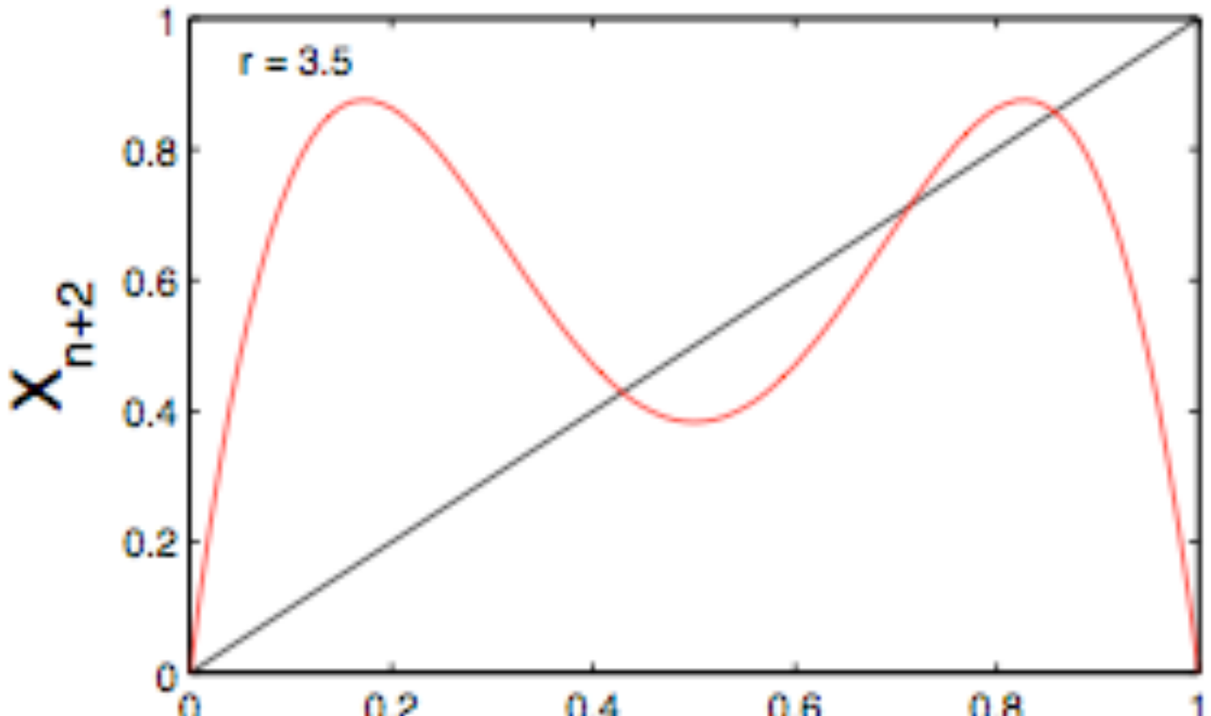




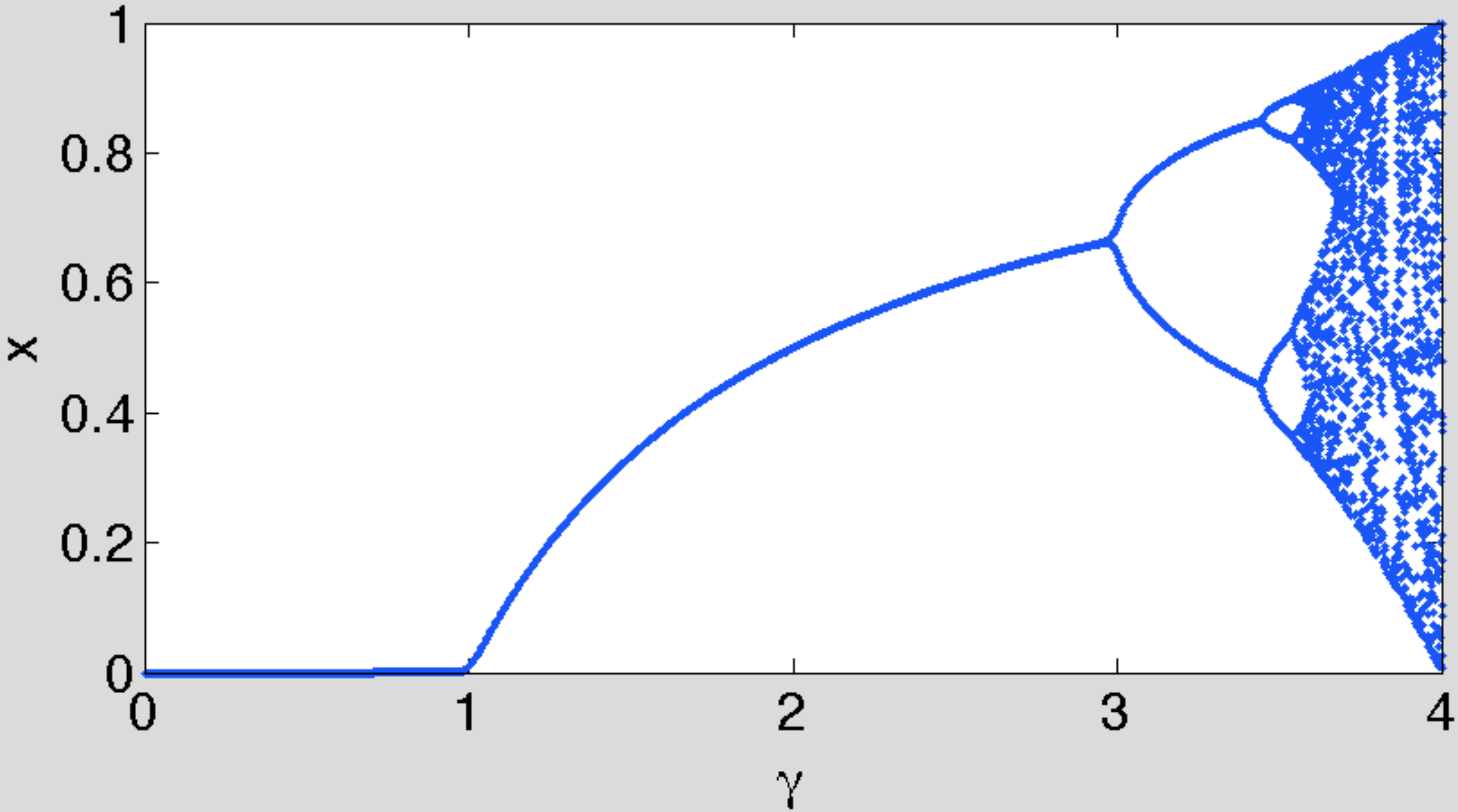
Fixed-point method: Logistic equation



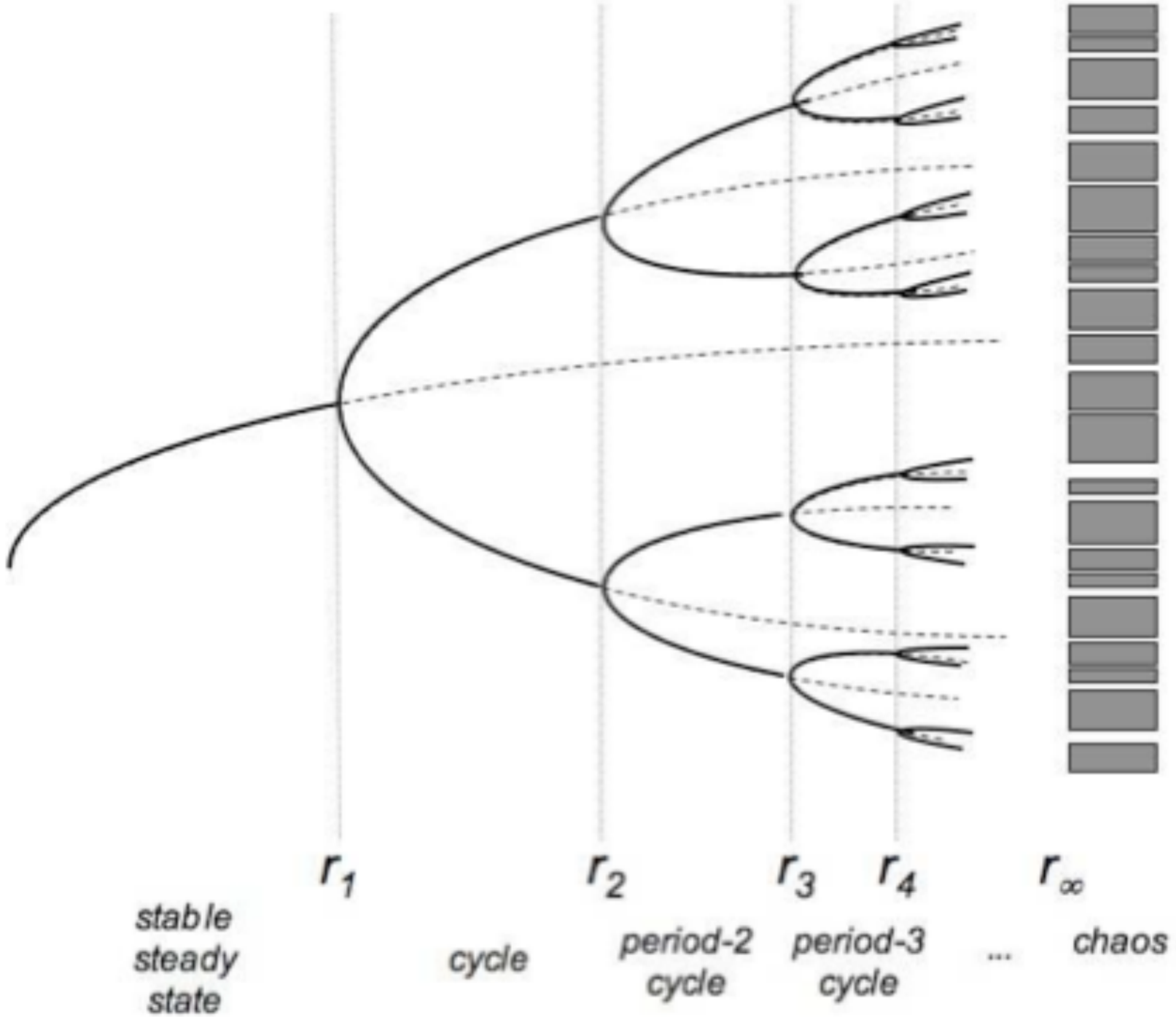




Logistic equation: Bifurcation



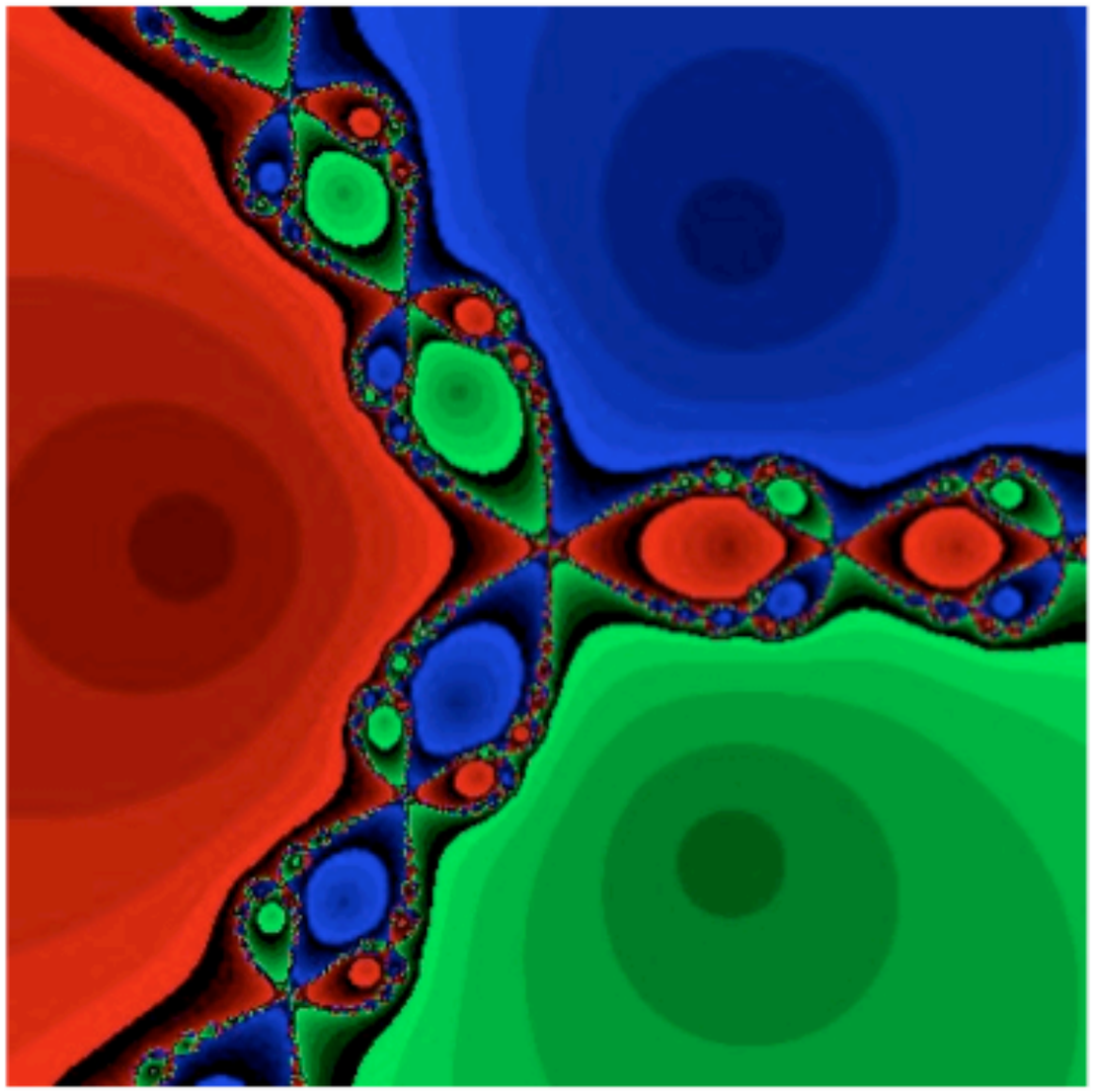
Logistic equations: Chaos



Root of equation: Newtown's Basins

Example: $x^3 + 1 = 0$

Newton-Raphson cont.



Red:
x= -1

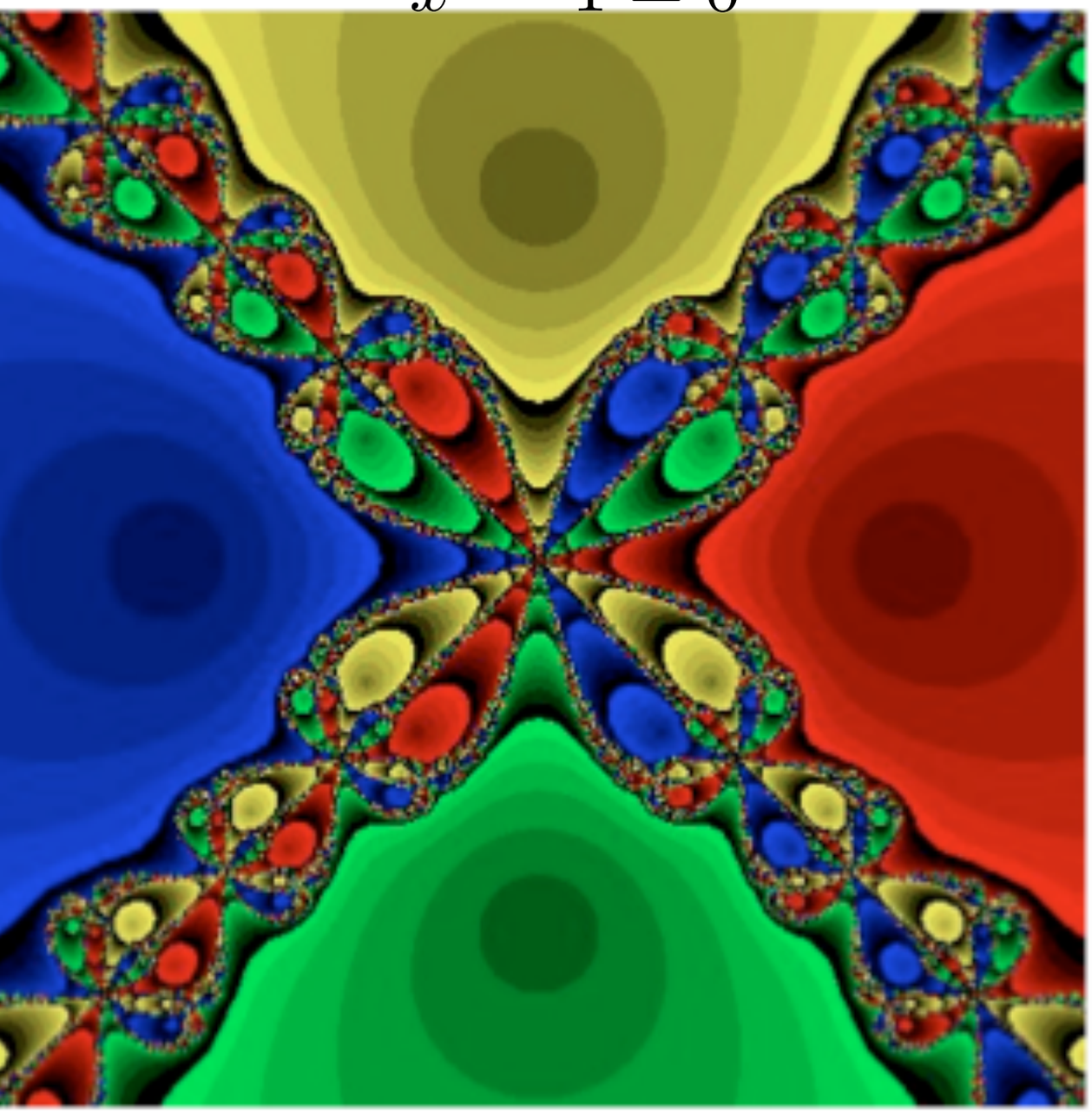
Blue:
x=0.5 + 0.866i

Green:
x=0.5 - 0.866i

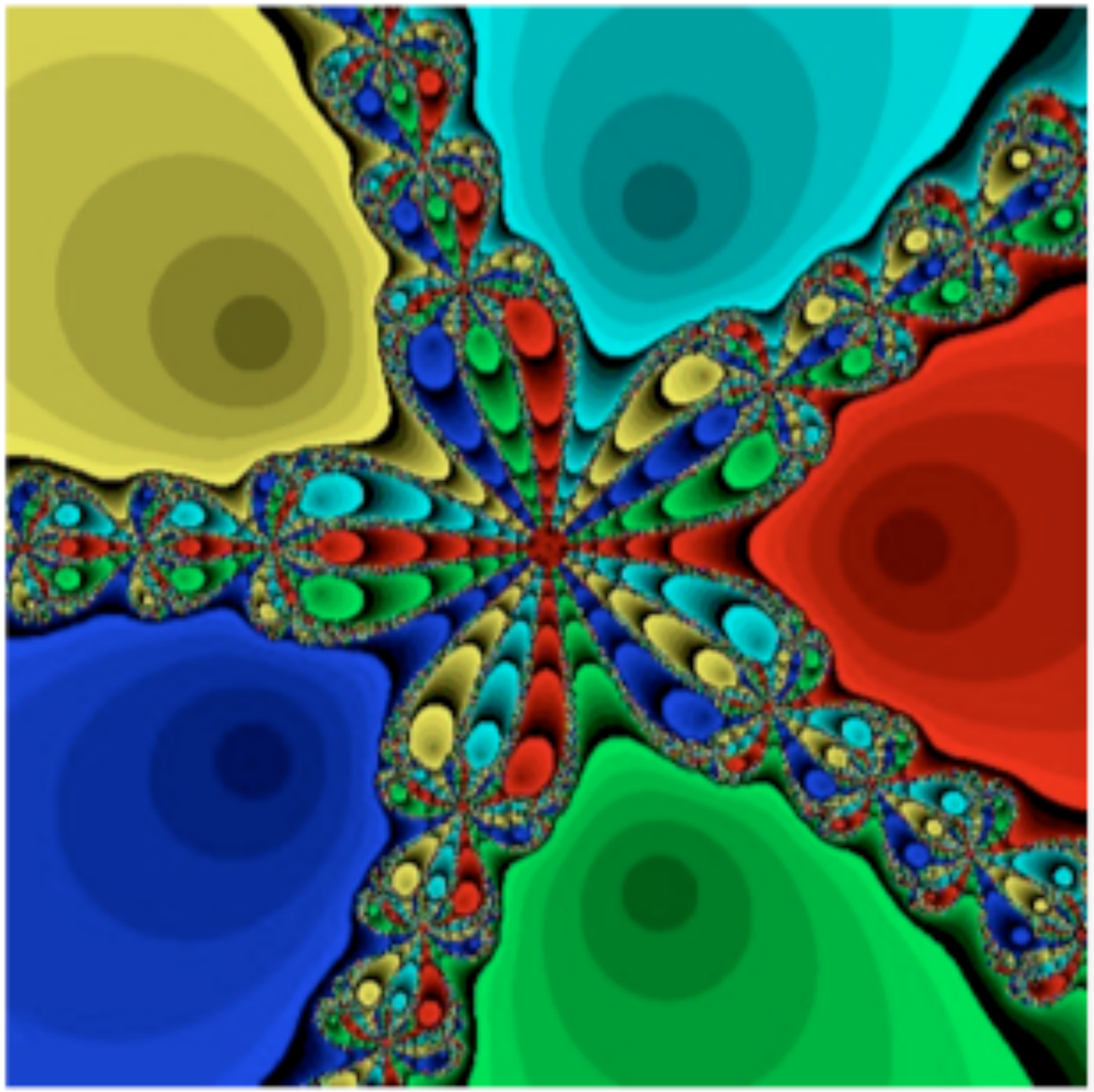
Attractors in Newtown's Basins

Example:

$$x^4 - 1 = 0$$



$$x^5 - 1 = 0$$



Applied Mathematics and Computation 215 (2009) 1084–1090

Contents lists available at ScienceDirect

Applied Mathematics and Computation

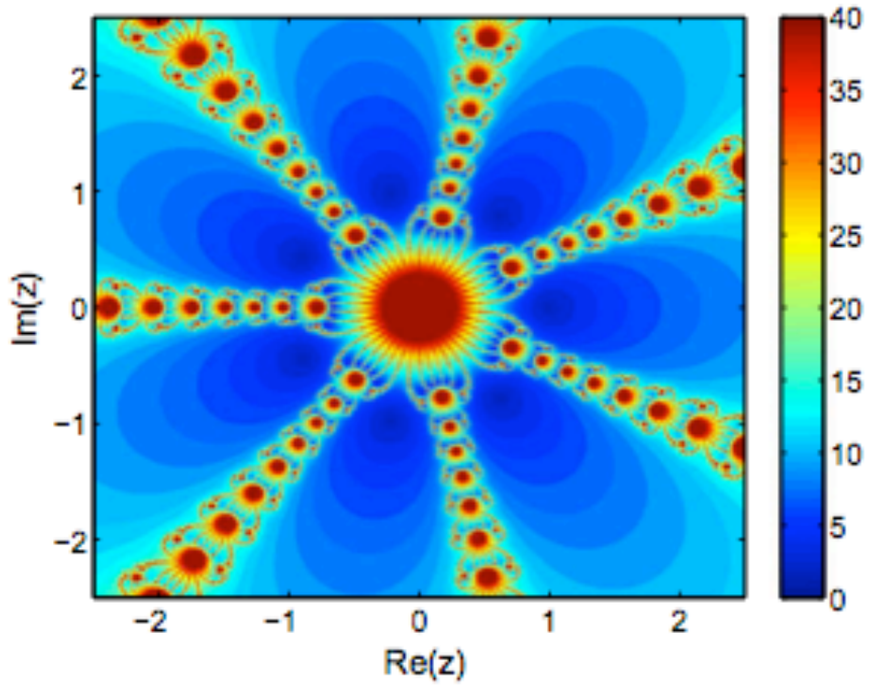
journal homepage: www.elsevier.com/locate/amc

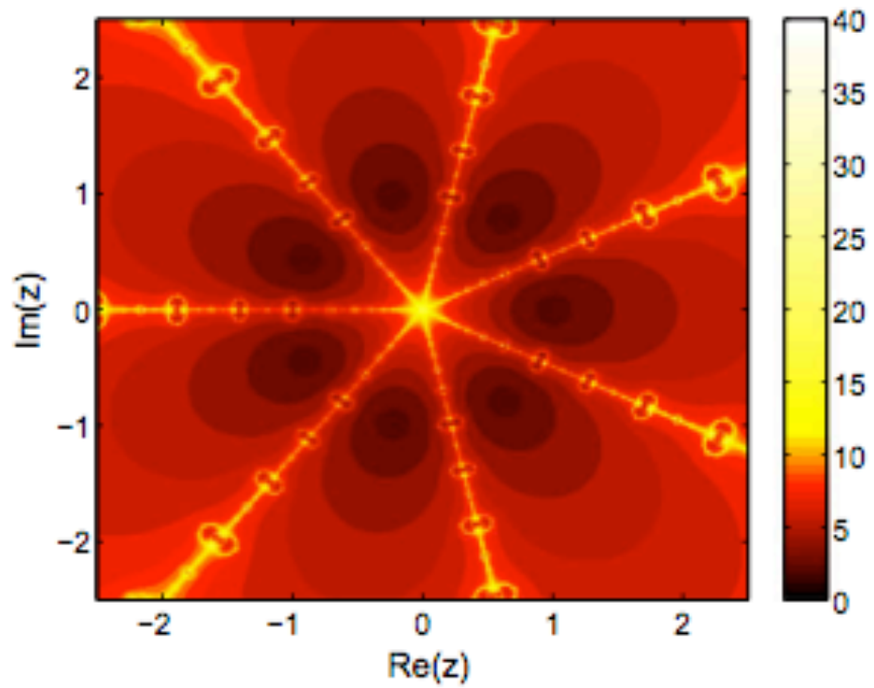
Newton's method's basins of attraction revisited

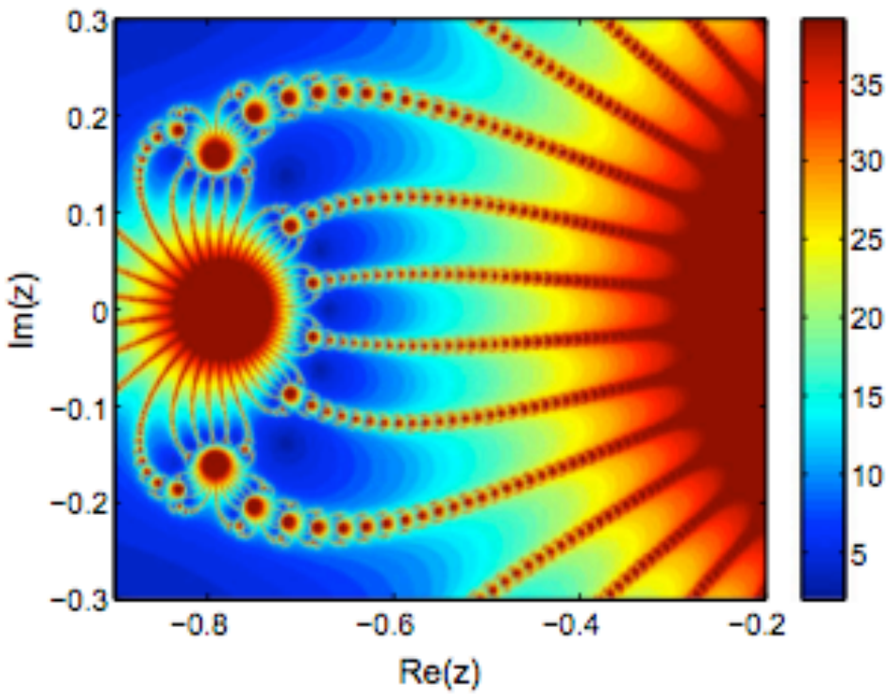
H. Susanto [a], N. Karjanto [b,*]

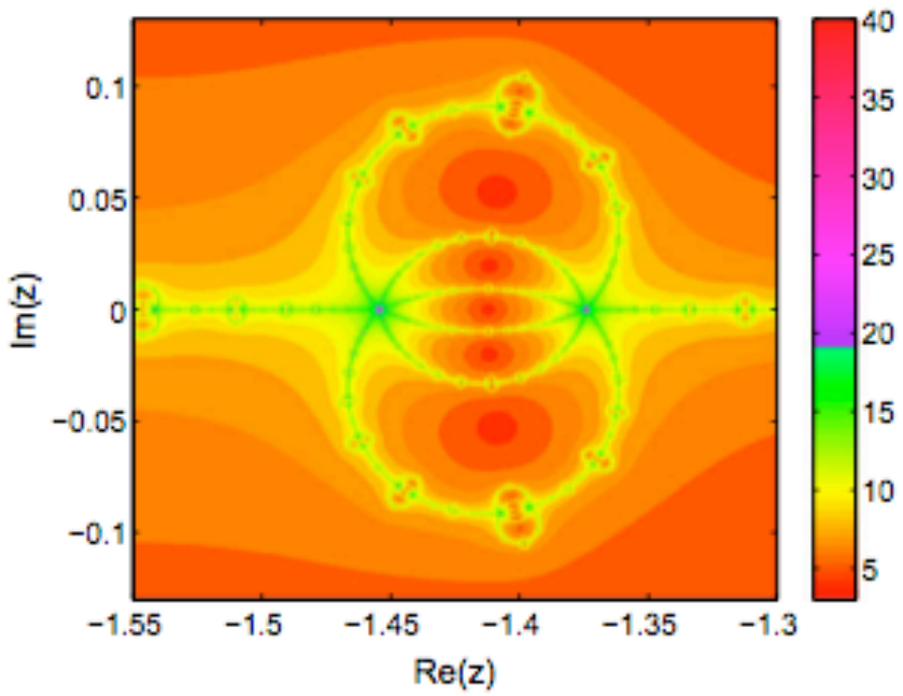
[a] School of Mathematical Sciences, University of Nottingham, University Park, Nottingham NG7 2RD, UK
[b] Department of Applied Mathematics, Faculty of Engineering, The University of Nottingham Malaysia Campus, Semenyih 43500, Selangor, Malaysia

Newton's method's Basins

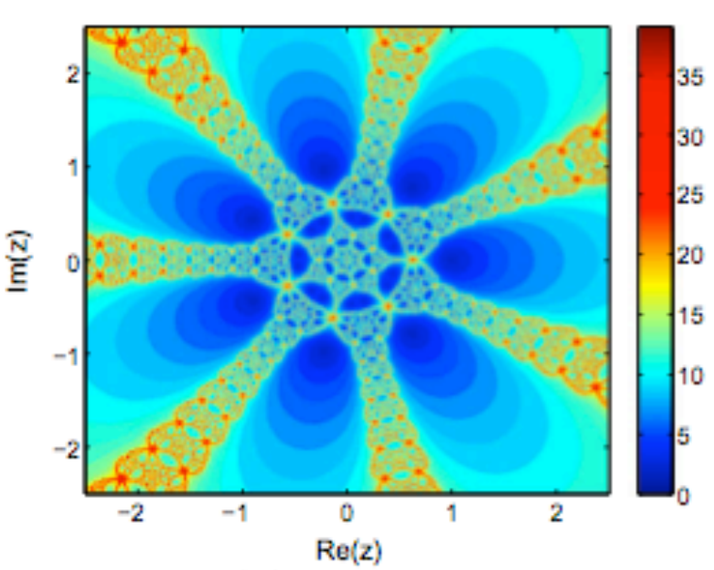




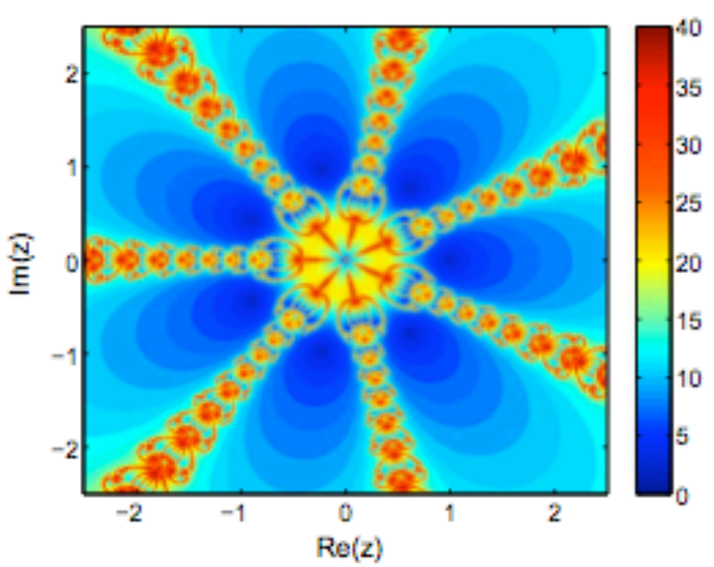




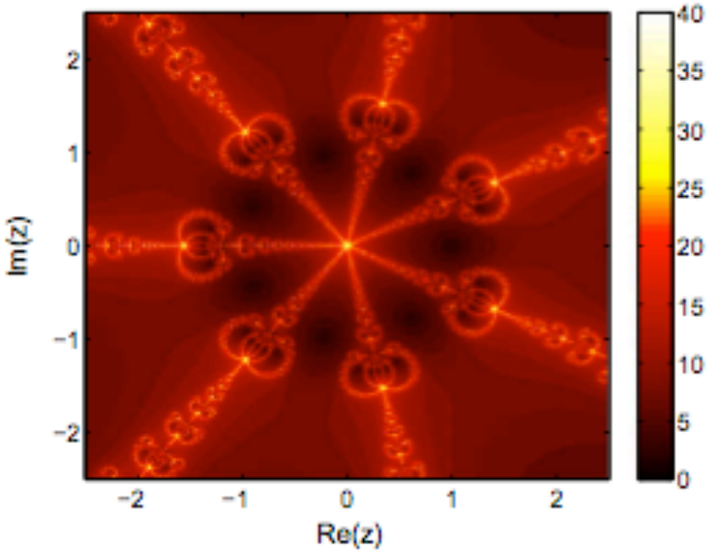
$$f(z) = z^7 - 1$$



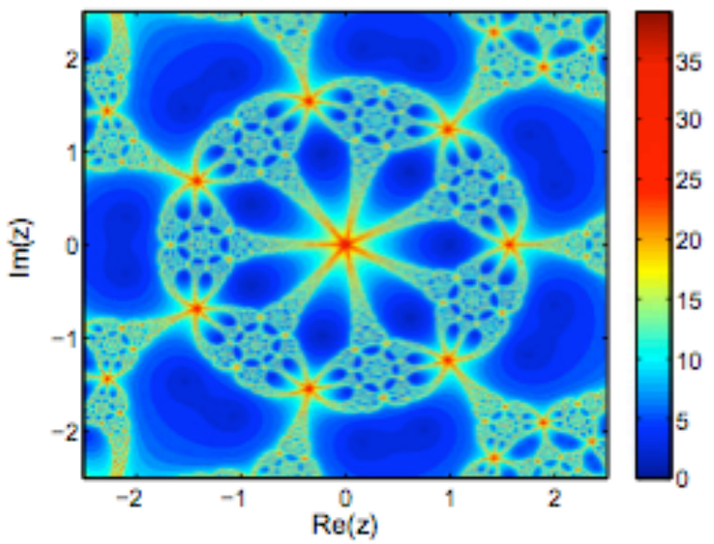
(a) $a_1 = 0.05$



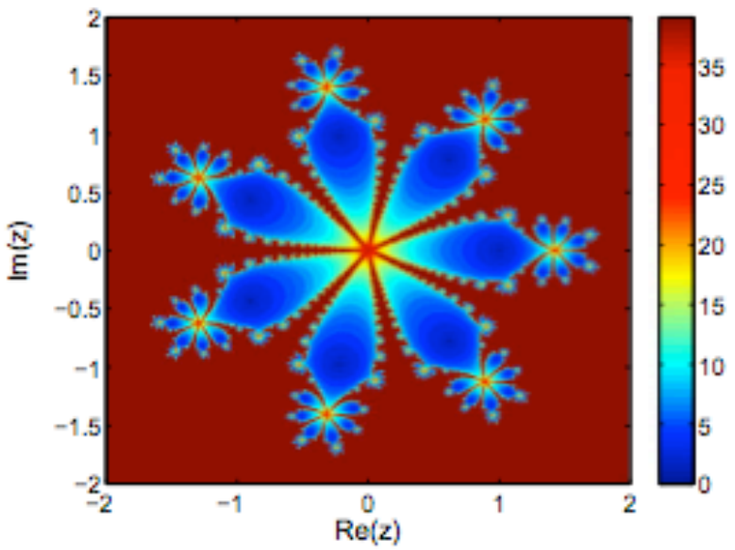
(b) $a_1 = -0.005$



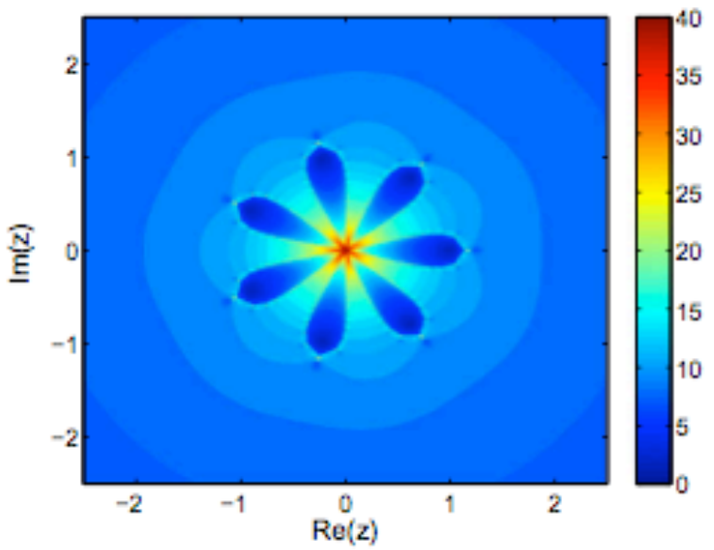
(c) $a_1 = -0.95$



(d) $a_1 = -1.05$



(e) $a_1 = -1.1$



(f) $a_1 = -1.5$

$$f(z) = z^7 - 1$$

$$z_{n+1} = F(z_n) = z_n - \frac{f(z_n)f'(z_n)}{a_0(f'(z_n))^2 + a_1 f(z_n)f''(z_n)}.$$

Roots of equation: Newton's method

Taylor expansion to the 1st-order:

$$f(x) = f(x_0) + \left.\frac{df(x)}{dx}\right|_{x=x_0} (x - x_0) + \mathbf{O}(x)$$

- Newtwon's method:

$$\begin{aligned} x_{\text{new}} &= x_{\text{old}} + \Delta x, \\ &= x_{\text{old}} - \frac{f(x_0)}{f'(x_0)} \end{aligned}$$

Roots of equation: Secant method

- Finite difference approximation:

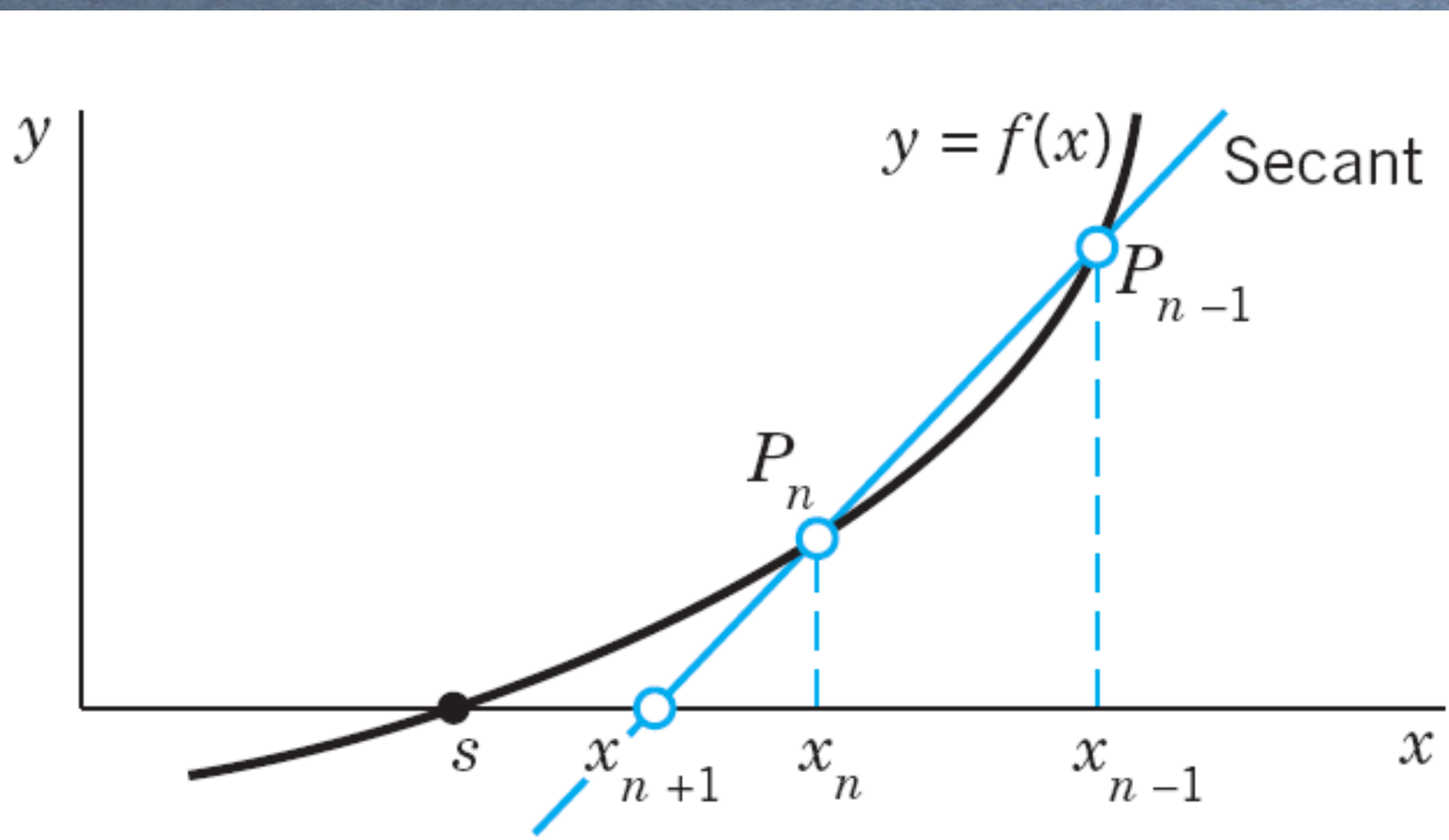
$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} + \mathbf{O}(\delta x^2)$$

- secant method:

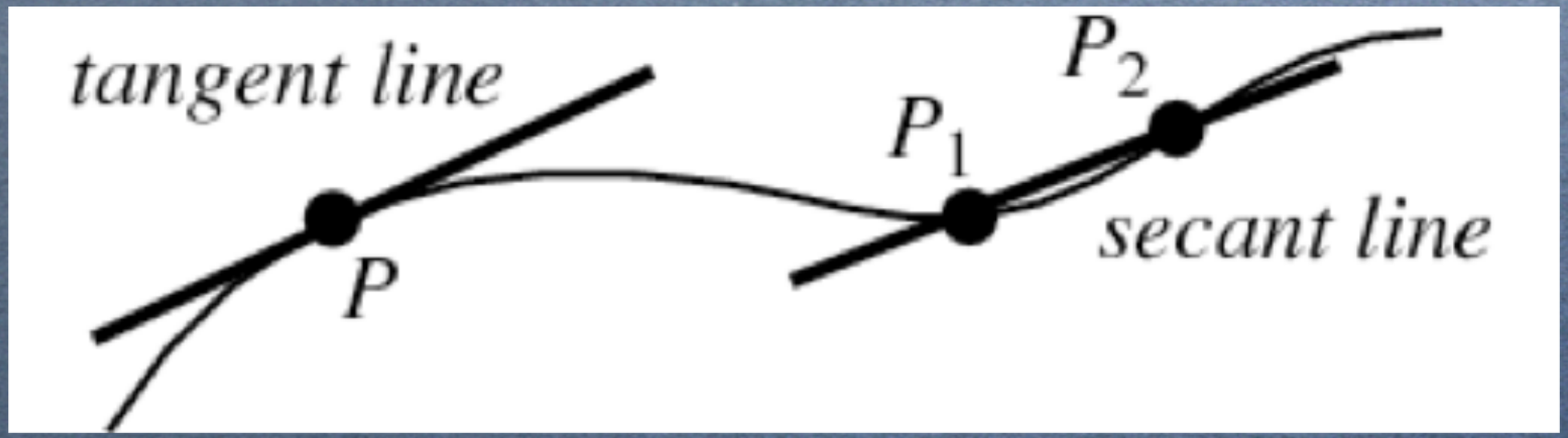
$$\begin{aligned} x_{\text{new}} &= x_{\text{old}} + \Delta x, \\ &= x_{\text{old}} - [\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}] f(x_0) \end{aligned}$$

Roots of equation: Secant method

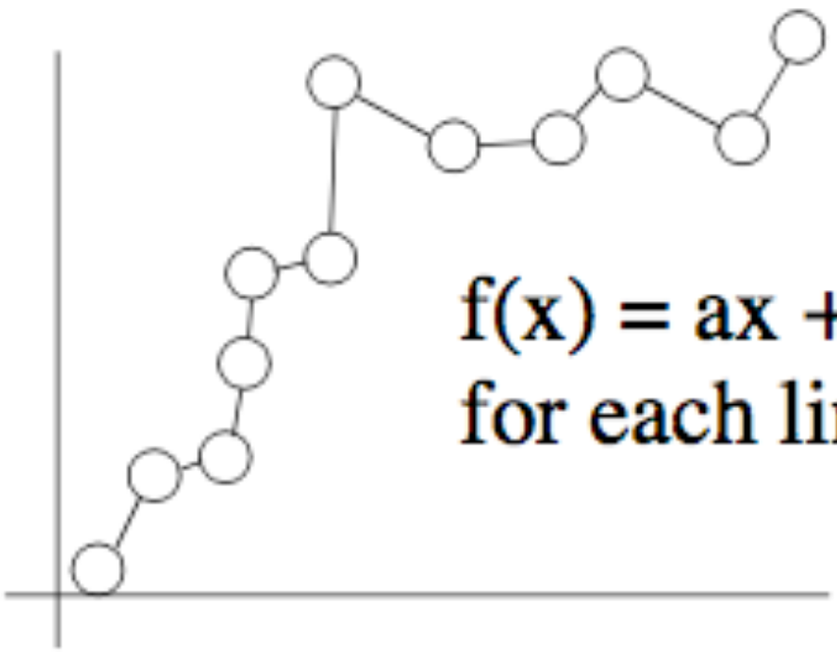
$$x_{n+1} = x_n - [\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}] f(x_0)$$



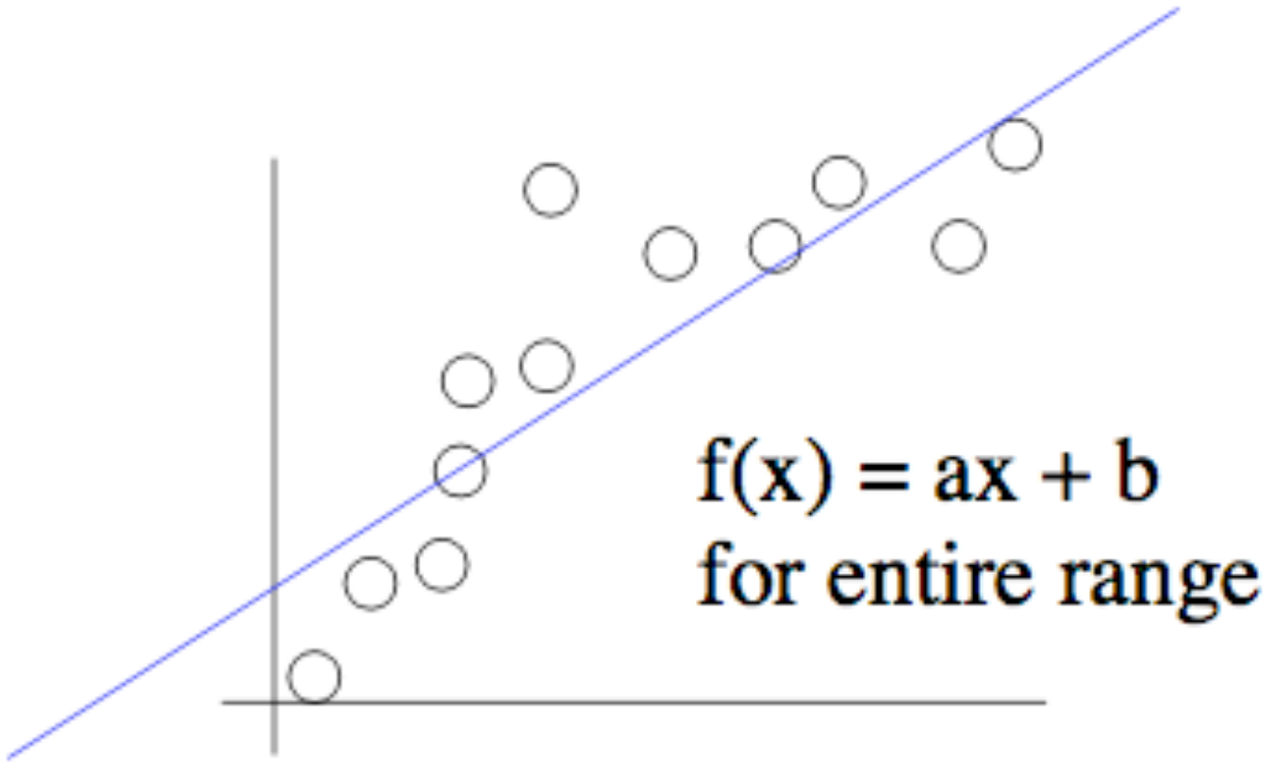
Roots of equation: Secant line



Interpolation:



Interpolation



Curve Fitting

Interpolation: Polynomial interpolation

☐ Give two points:

$$(x_0, y_0) \quad \text{and} \quad (x_1, y_1),$$

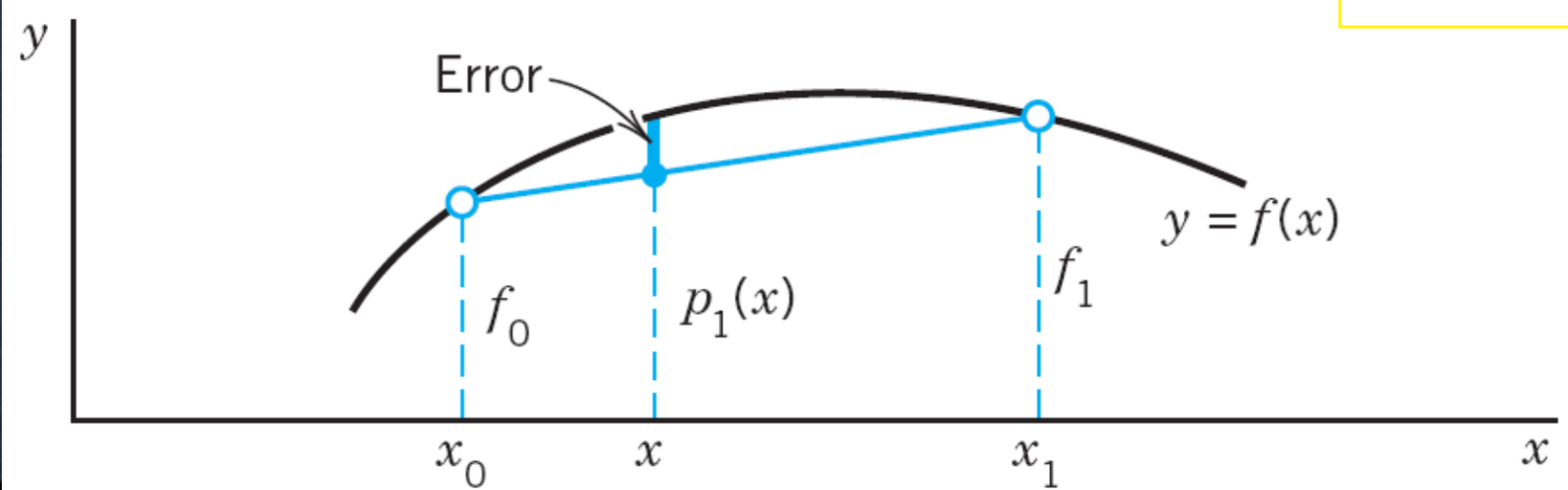
the polynomial function $p(x)$ satisfying

$$\begin{aligned} p(x_0) &= y_0; \\ p(x_1) &= y_1; \end{aligned}$$

Linear interpolation:

is

$$p(x) = (\frac{x - x_1}{x_0 - x_1})y_0 + (\frac{x - x_0}{x_1 - x_0})y_1.$$



Interpolation: Polynomial interpolation

For a given set of $N+1$ dat points

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)\},$$

we want to find the coefficients of an Nth-degree polynomial function to match them:

$$\boxed{P_N(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N,}$$

The coefficients can be obtained by solving the following system of linear equations,

$$\begin{aligned}
a_0 + x_0 a_1 + x_0^2 a_2 + \cdots + x_0^N a_N &= y_0, \\
a_0 + x_1 a_1 + x_1^2 a_2 + \cdots + x_1^N a_N &= y_1, \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\
a_0 + x_N a_1 + x_N^2 a_2 + \cdots + x_N^N a_N &= y_N.
\end{aligned}$$

Interpolation: Lagrange interpolation

Lagrange Polynomial:

$$LP_N(x) = y_0 \frac{(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_N)} + y_1 \frac{(x-x_0)(x-x_2)\cdots(x-x_N)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_N)} + \cdots + y_N \frac{(x-x_0)(x-x_1)\cdots(x-x_{N-1})}{(x_N-x_0)(x_N-x_1)\cdots(x_N-x_{N-1})}$$

or

$$LP_N(x) = \sum_{m=0}^{N} y_m L_{N,m}(x), \quad \text{with} \quad L_{N,m} = \prod_{k \neq m}^{N} \frac{x-x_k}{x_m-x_k}$$

Homework 2: Lagrange interpolation

Give the points:

$$(0,0), \quad (1,1), \quad (2,2.001), \quad (3,3), \quad (4,4), \quad (5,5)$$

1. Find the Lagrange polynomial that match all of these points.

2. Evaluate the polynomial function at $x = 20$ for the *extrapolation*, and is there anything wrong for the value at $x = 20$?

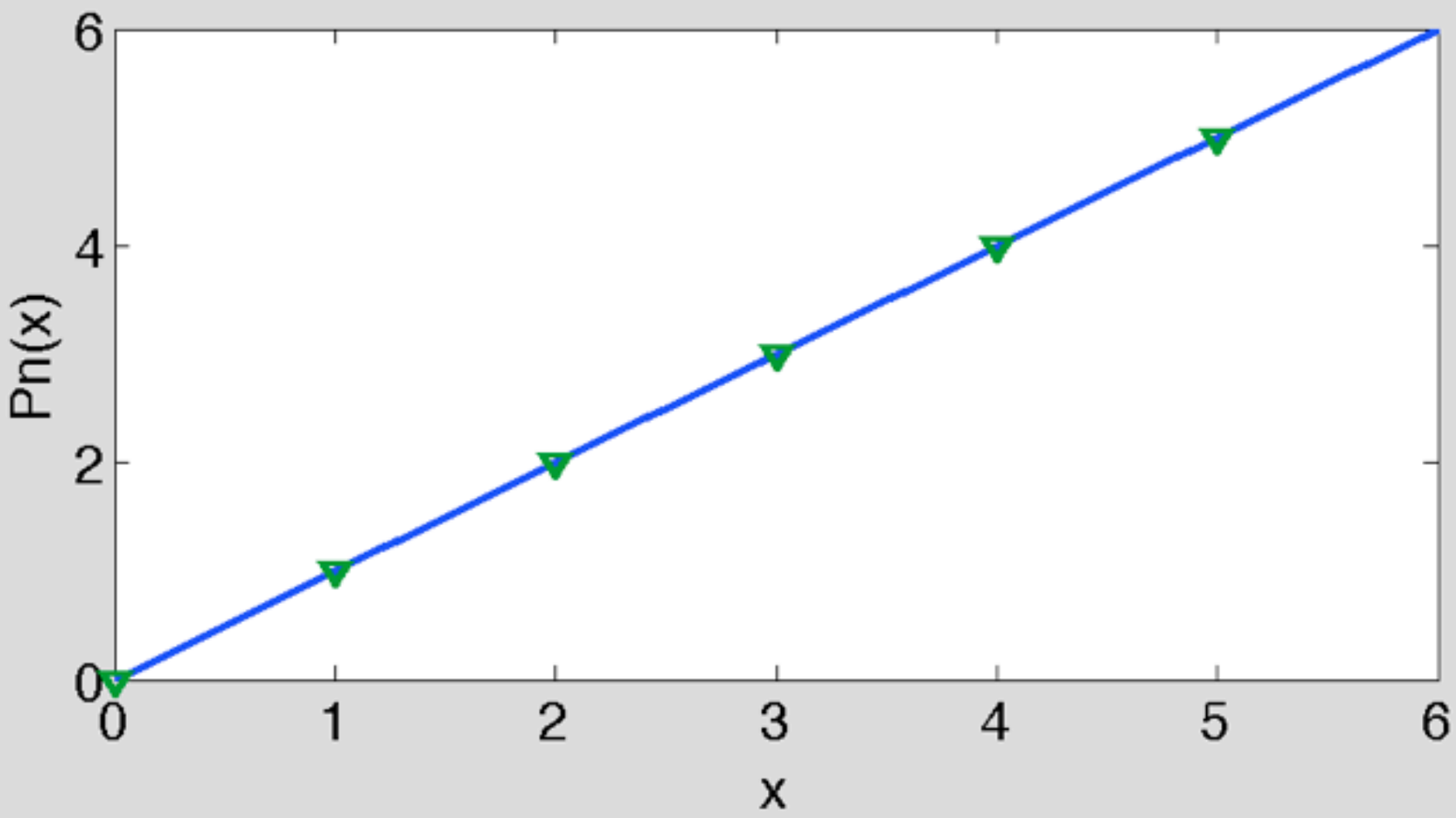
Homework 2: Lagrange interpolation

- The Lagrange polynomial is

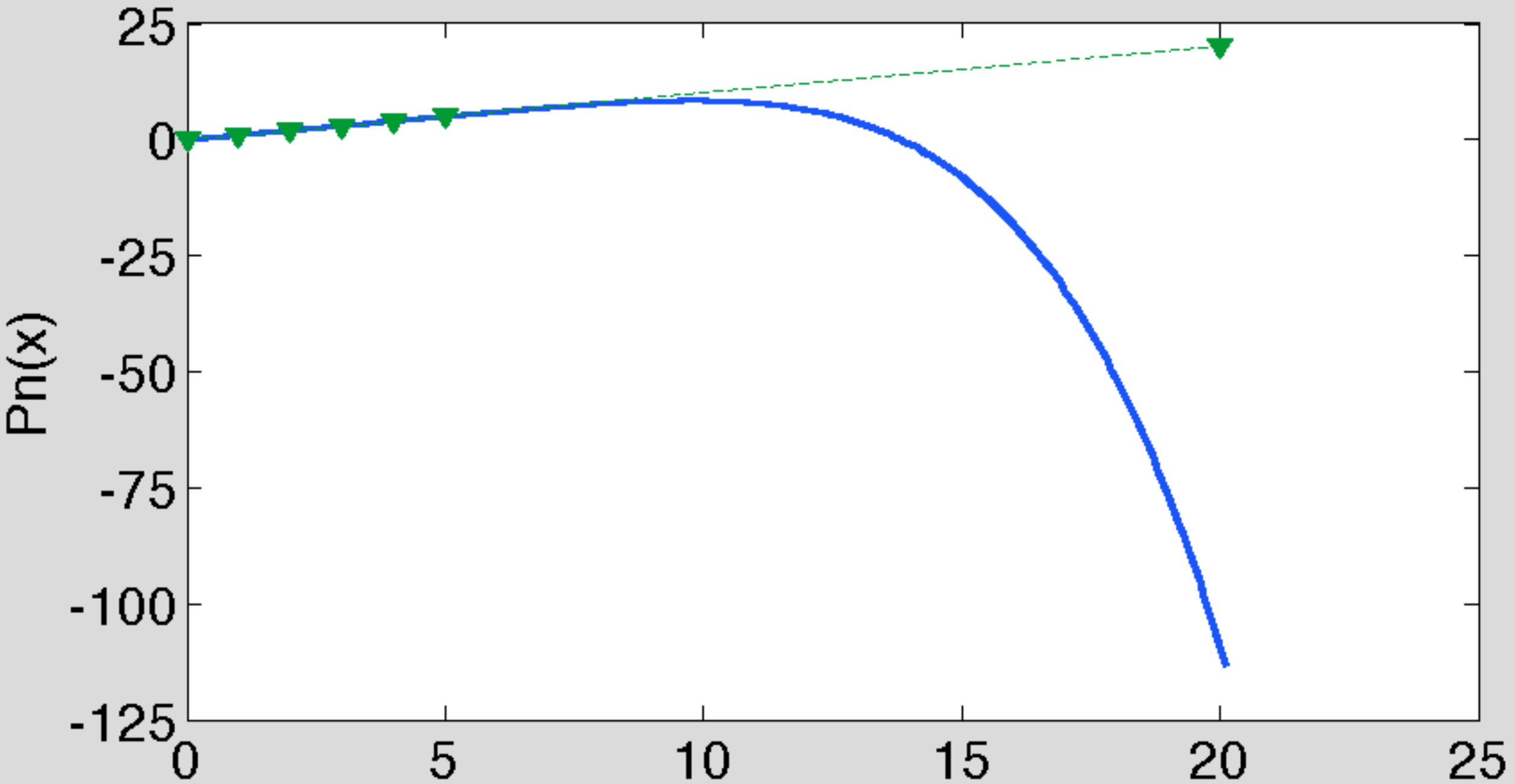
$$Pn(x) = 0.995\,x + 0.00891667\,x^2 - 0.00491667\,x^3 + 0.00108333\,x^4 - 0.0000833333\,x^5.$$

- For $x = 20$,

$$Pn(20) = -109.2, \quad \text{far away from} \quad 20.$$



Homework 2: Lagrange interpolation



Homework 2: Lagrange interpolation

Give the 17 points:

$$(-1.0000, 0.0588) \quad (-0.8750, 0.0755) \quad (-0.7500, 0.1) \quad (-0.6250, 0.1379)$$
$$(-0.5000, 0.2) \quad (-0.3750, 0.3077) \quad (-0.2500, 0.5) \quad (-0.1250, 0.8)$$
$$(0, 1.0) \quad (0.1250, 0.8) \quad (0.2500, 0.5) \quad (0.3750, 0.3077)$$
$$(0.5000, 0.2) \quad (0.6250, 0.1379) \quad (0.7500, 0.1) \quad (0.8750, 0.0755) \quad (1.0000, 0.0588).$$

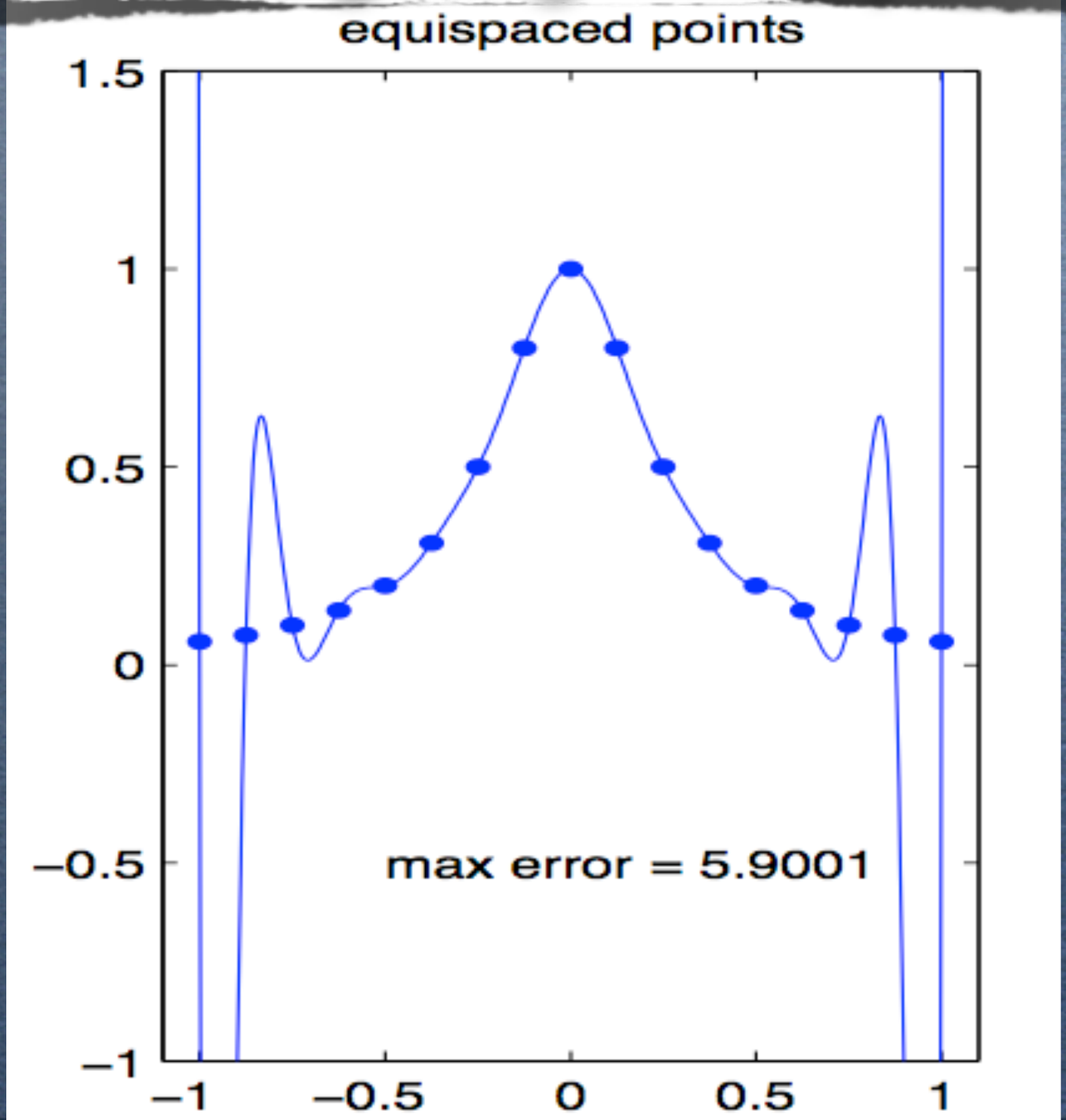
1. Find the Lagrange polynomial, $LP(x)$, that matches all of these points.
2. Compare your polynomial function with respect to the function

$$f(x) = \frac{1}{1 + 16x^2},$$

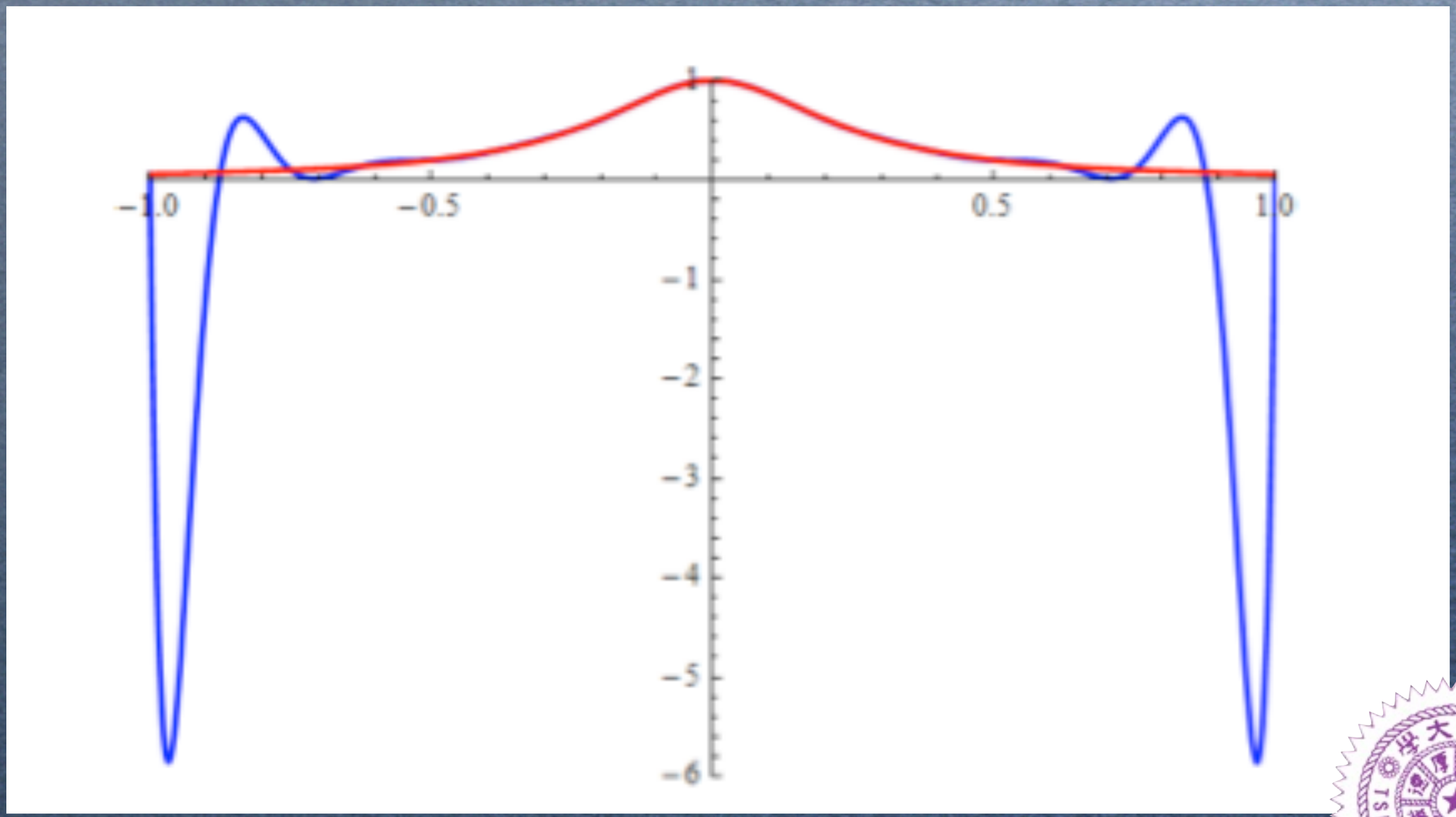
and evaluate the difference (error) between them by calculating

$$\text{error} = \int_{-1}^{1} |LP(x) - f(x)|^2 \, \mathrm{d}x$$

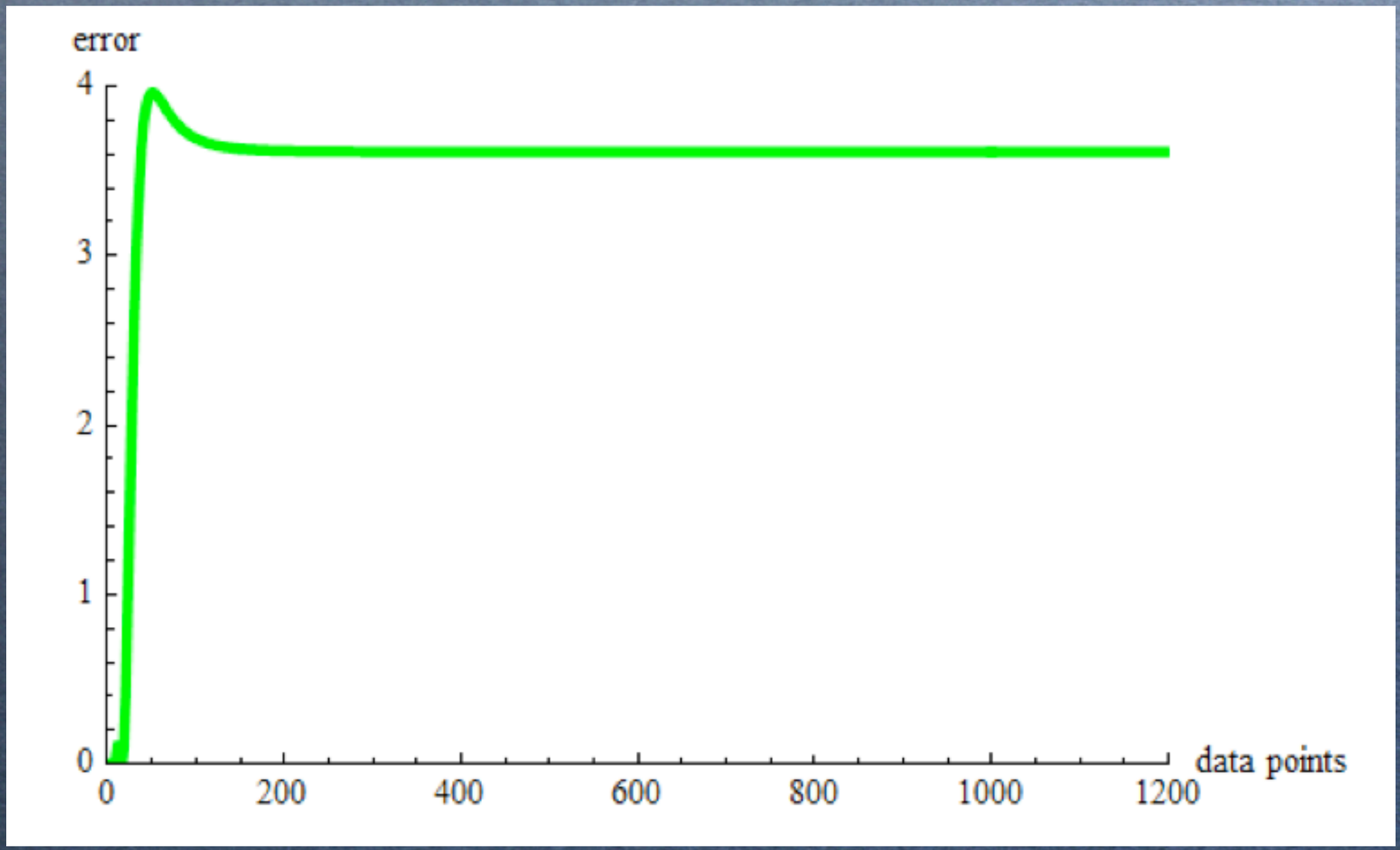
Homework 2: Lagrange interpolation



Homework 2: Lagrange interpolation

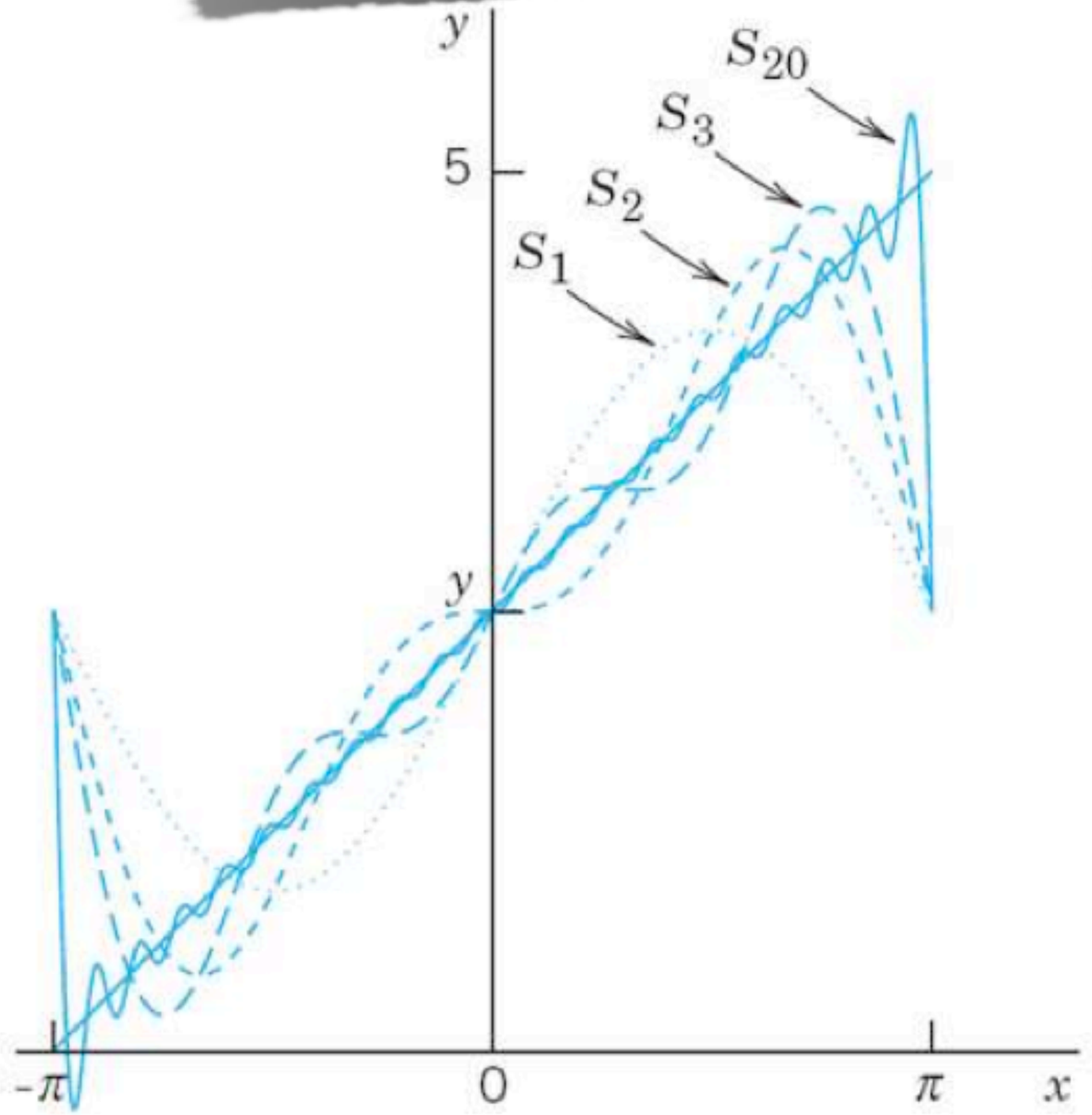


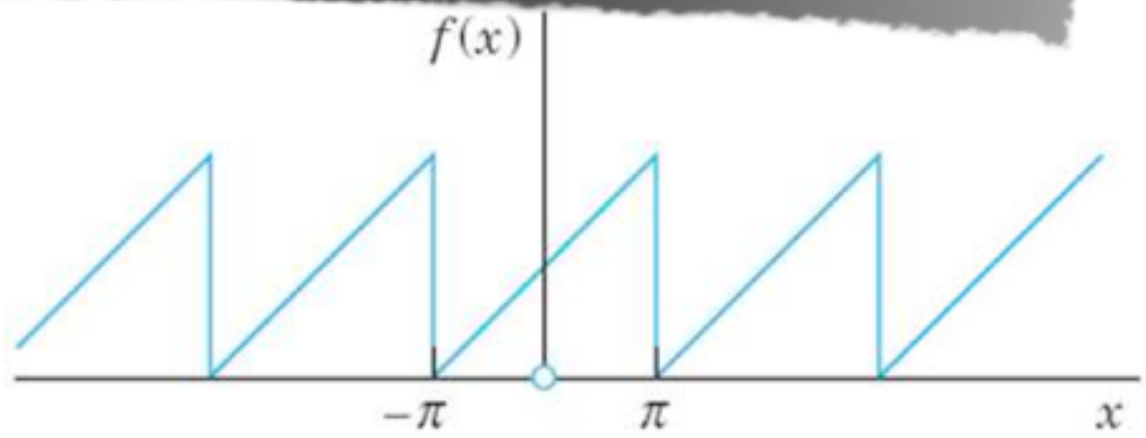
Homework 2: Lagrange interpolation



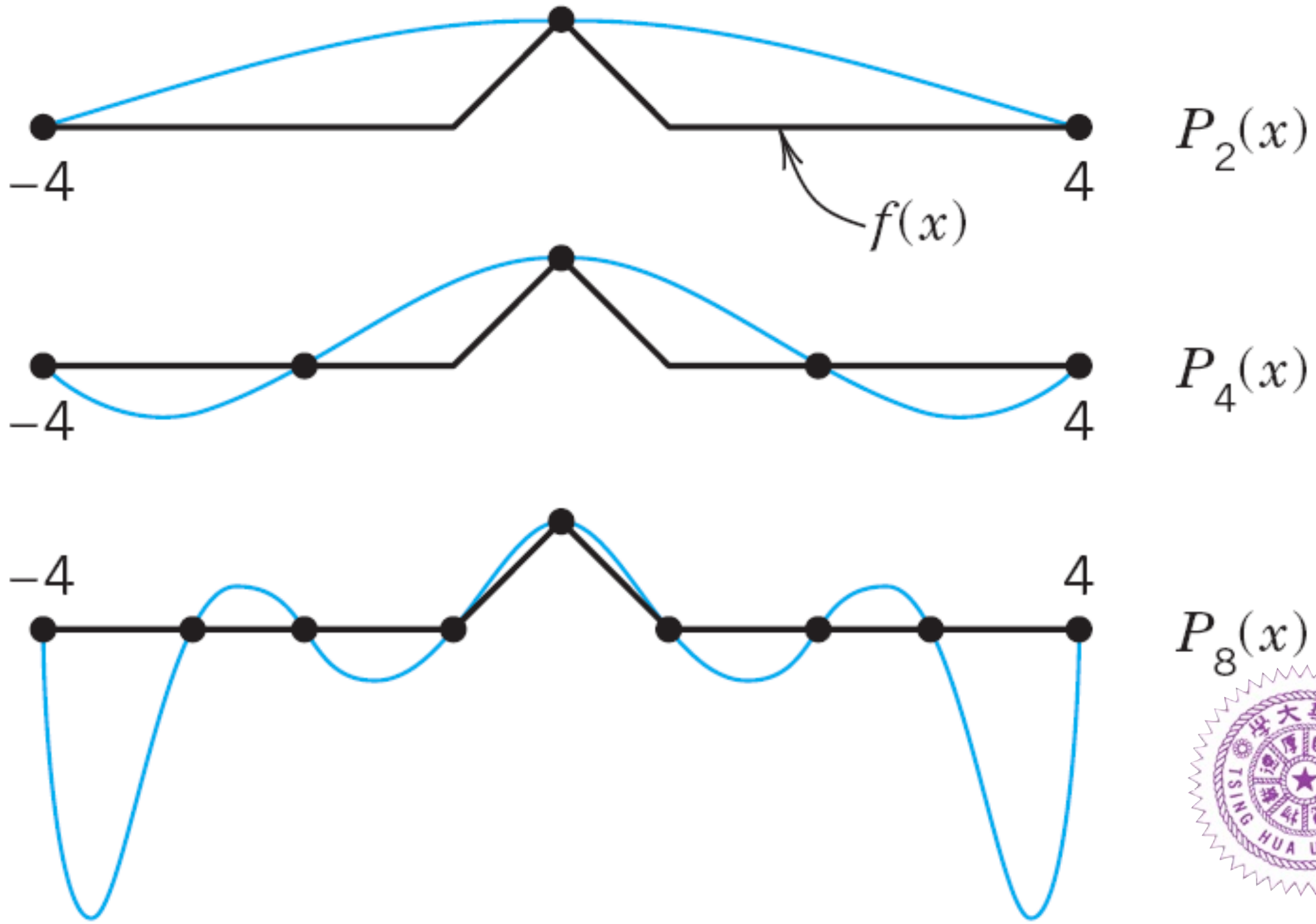
3.61225

Fourier Series: Gibbs phenomenon

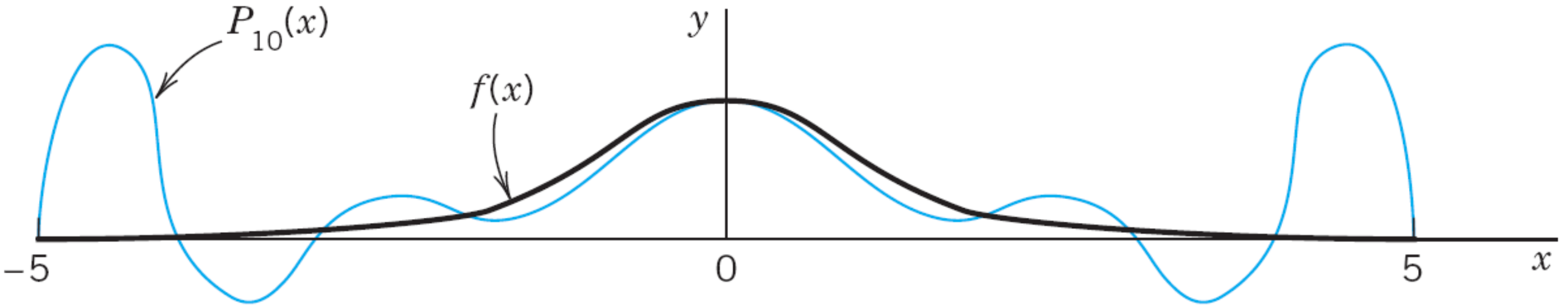




Polynomial wiggle: Runge phenomenon



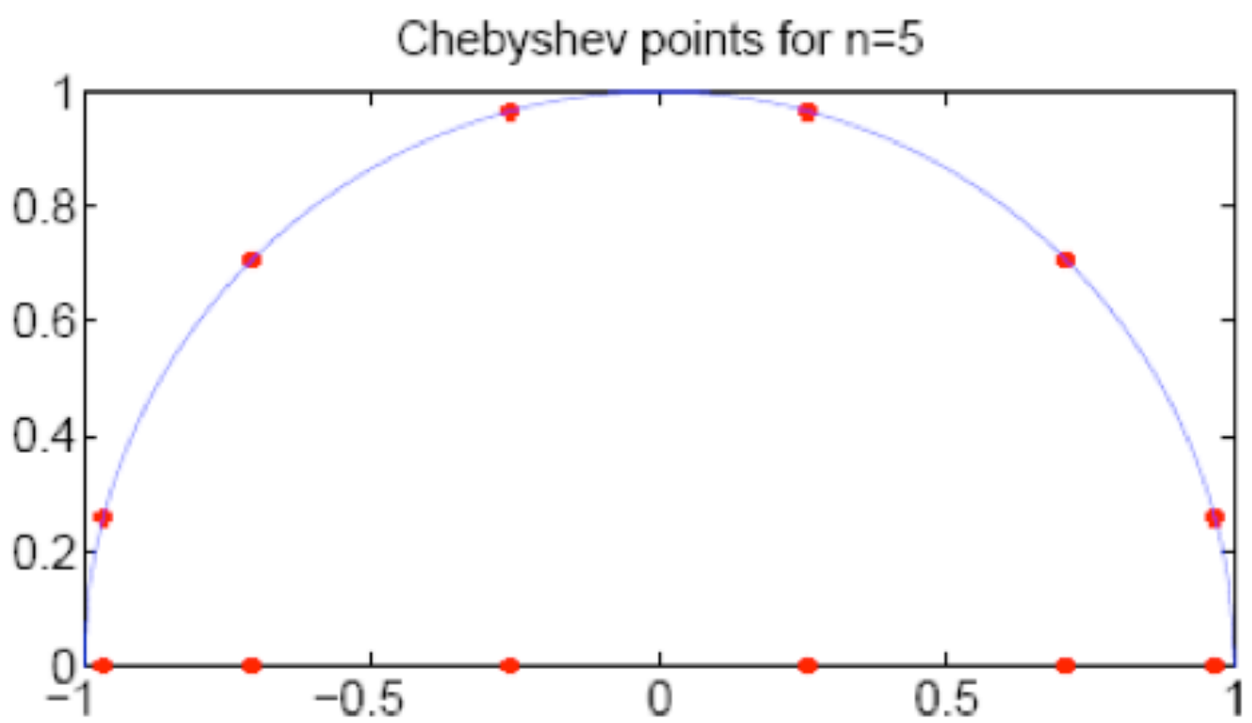
Interpolation: Runge phenomenon

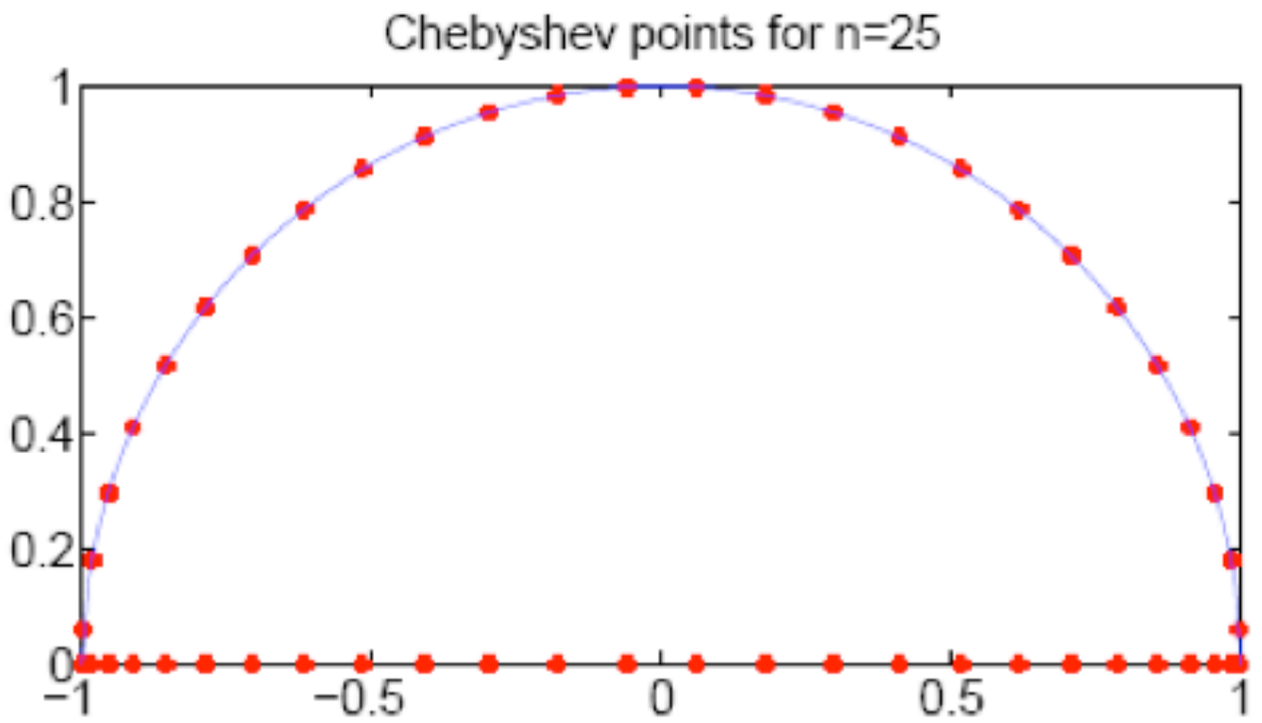


Interpolation: Chebyshev points

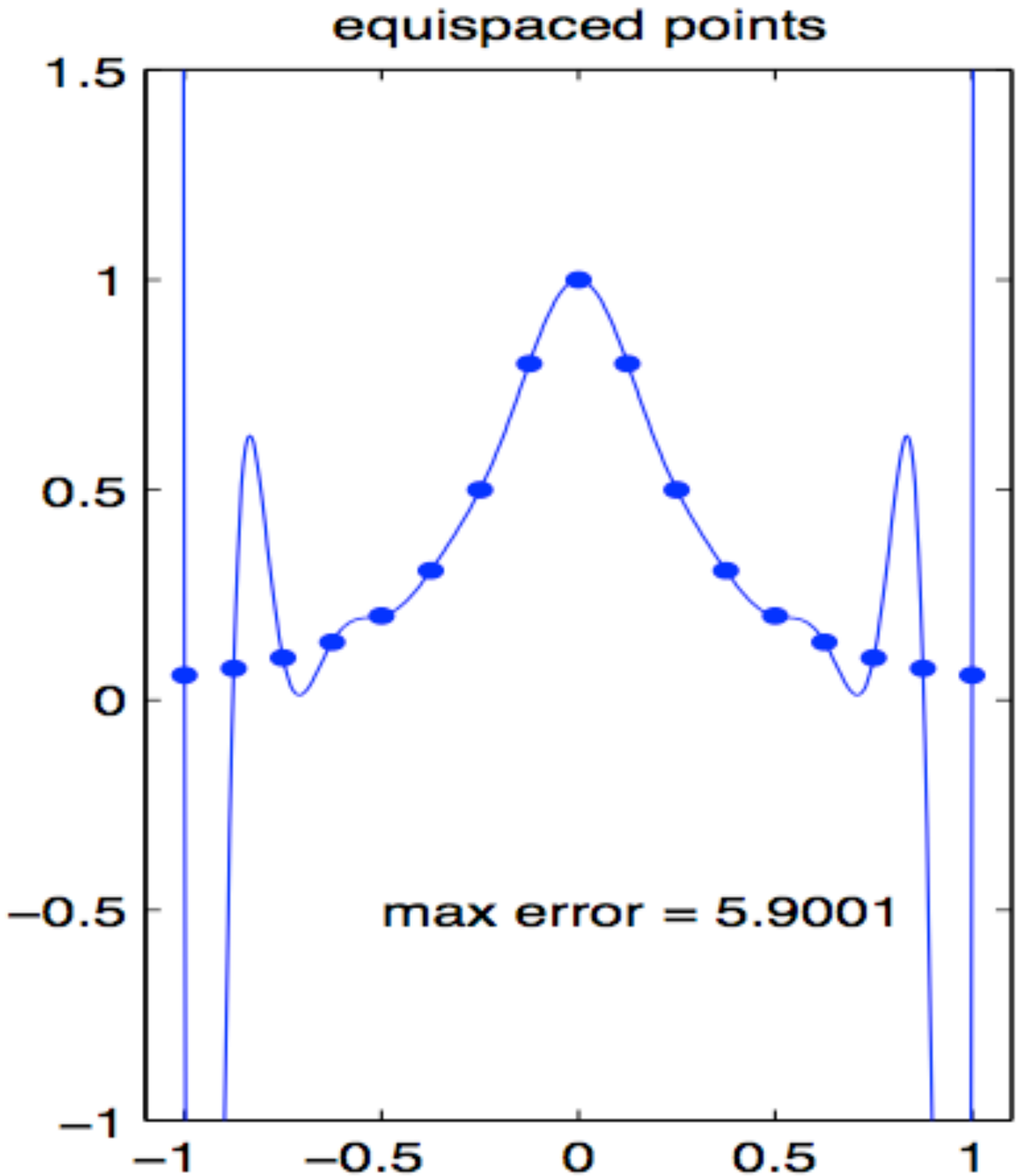
- For $n+1$ nodes in $[-1,1]$, the corresponding Chebyshev notes are

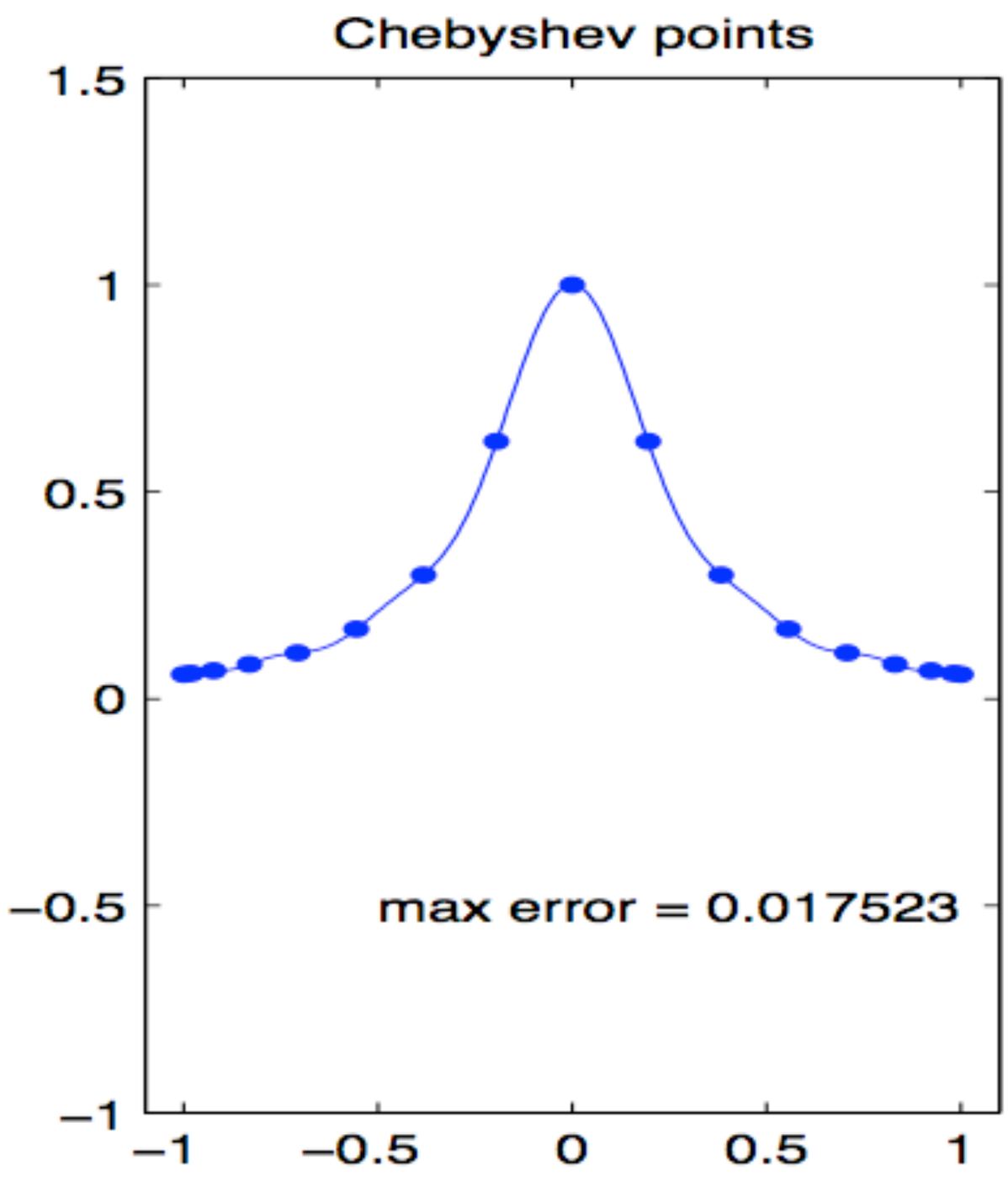
$$x_i = \mathrm{Cos}\,[\frac{2\,i+1}{2\,n+2}\pi], \qquad 0 \le i \le n.$$





Interpolation: Runge phenomenon





Homework 3: Chebyshev nodes

Give the 17 points:

$(-0.995734, 0.0592987)$ $(-0.961826, 0.0632842)$ $(-0.895163, 0.0723533)$ $(-0.798017, 0.0893711)$
$(-0.673696, 0.121038)$ $(-0.526432, 0.184023)$ $(-0.361242, 0.323842)$ $(-0.18375, 0.649257)$
$(0, 1.0)$ $(0.18375, 0.649257)$ $(0.361242, 0.323842)$ $(0.526432, 0.184023)$
$(0.673696, 0.121038)$ $(0.798017, 0.0893711)$ $(0.895163, 0.0723533)$ $(0.9618260.0632842)$
$(0.9957340.0592987)$

1. Find the Lagrange polynomial, $LP(x)$, that matches all of these points.
2. Compare your polynomial function with respect to the function

$$f(x) = \frac{1}{1 + 16x^2},$$

and evaluate the difference (error) between them by calculating

$$\text{error} = \int_{-1}^{1} |LP(x) - f(x)|^2 \, \mathrm{d}x$$

2D: Lagrange interpolation

- bilinear interpolation in 1D: given points $(x_{m-1}, f(x_{m-1}))$ and $(x_m, f(x_m))$,

$$f(x) = \frac{x - x_m}{x_{m-1} - x_m} f(x_{m-1}) + \frac{x - x_{m-1}}{x_m - x_{m-1}} f(x_m)$$

- bilinear interpolation in 2D:

$$\begin{aligned} z(x,y) &= \frac{y - y_n}{y_{n-1} - y_n} z(x, y_{n-1}) + \frac{y - y_{n-1}}{y_n - y_{n-1}} z(x, y_n) \\ &= \frac{y - y_n}{y_{n-1} - y_n} [\frac{x - x_n}{x_{n-1} - x_n} z(x_{n-1}, y_{n-1}) + \frac{x_n - x_{n-1}}{x_n - x_{n-1}} z(x_n, y_{n-1})] \\ &+ \frac{y - y_{n-1}}{y_n - y_{n-1}} [\frac{x - x_n}{x_{n-1} - x_n} z(x_{n-1}, y_n) + \frac{x - x_{n-1}}{x_n - x_n} z(x_n, y_{n-1})] \end{aligned}$$

Newton's divided difference Interpolation:

Give the 5 points:

$$(0,-5),(1,-3),(-1,-15),(2,39),(-2,9)$$

The interpolating polynomial is constructed by Newton algorithm:

0. For (x_0,y_0), the polynomial $P_0(x)=y_0=-5$.
1. For (x_1,y_1), the polynomial $P_1(x)$ is

$$p_1(x)=P_0(x)+c_1\,(x-x_0)=-5+c_1\,(x-0),$$

then $P_1(1)=-3$, we have $c_1=2$.

2. For (x_2,y_2), the polynomial $P_2(x)$ is

$$p_2(x)=P_1(x)+c_2\,(x-x_0)(x-x_1),$$

then $P_2(-1)=-15$, we have $c_2=-4$.

Newton's divided difference Interpolation:

3. For (x_3, y_3), the polynomial $P_3(x)$.

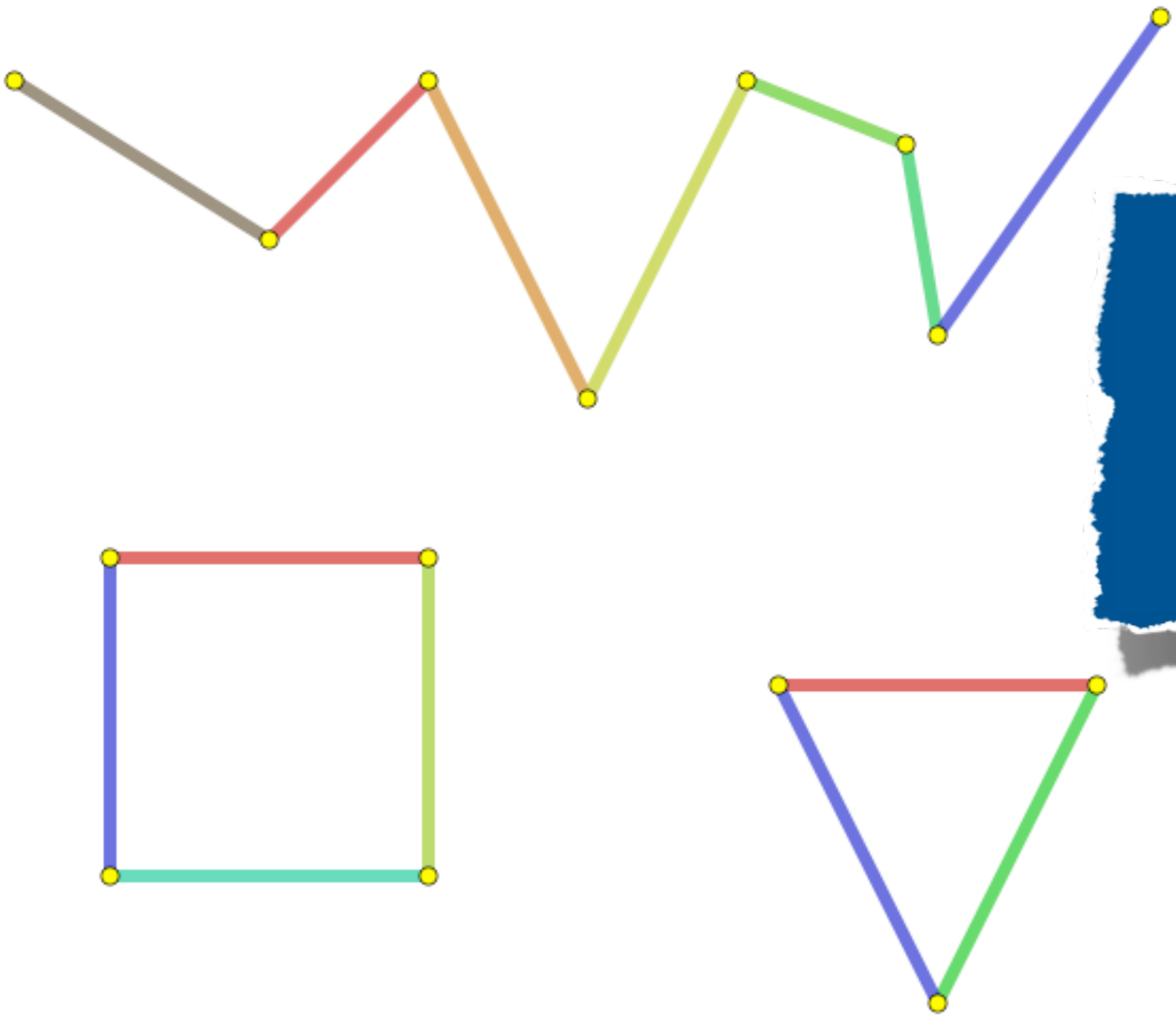
$$p_3(x) = P_2(x) + c_3\,(x - x_0)(x - x_1)(x - x_2),$$

then $P_3(2) = 39$, we have $c_3 = 8$.

4. For (x_4, y_4), we have

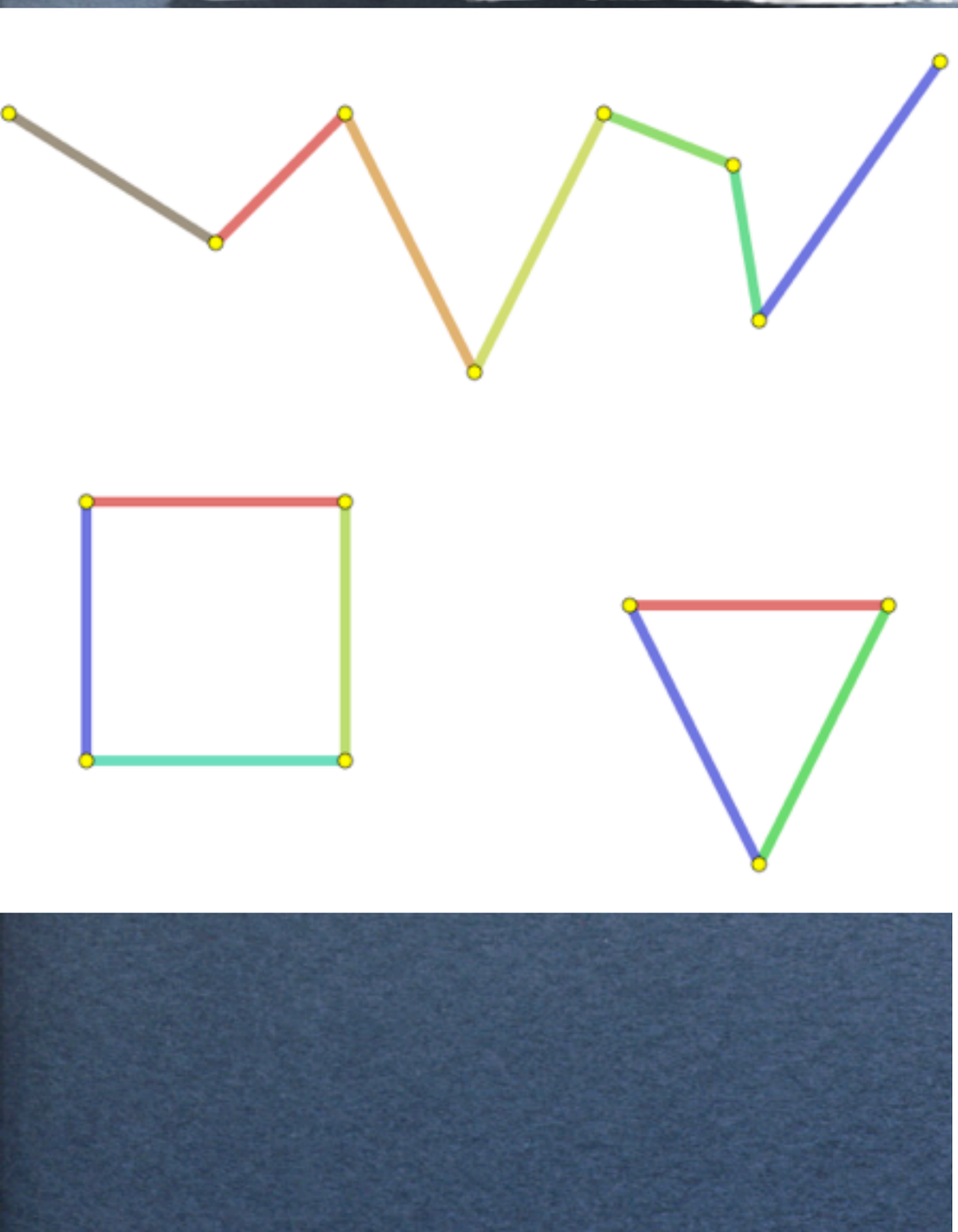
$$\begin{aligned} P_4(x) = \quad & -5 + 2(x - 0) - 4(x - 0)(x - 1) + 8(x - 0)(x - 1)(x + 1) \\ & +3(x - 0)(x - 1)(x + 1)(x - 2). \end{aligned}$$

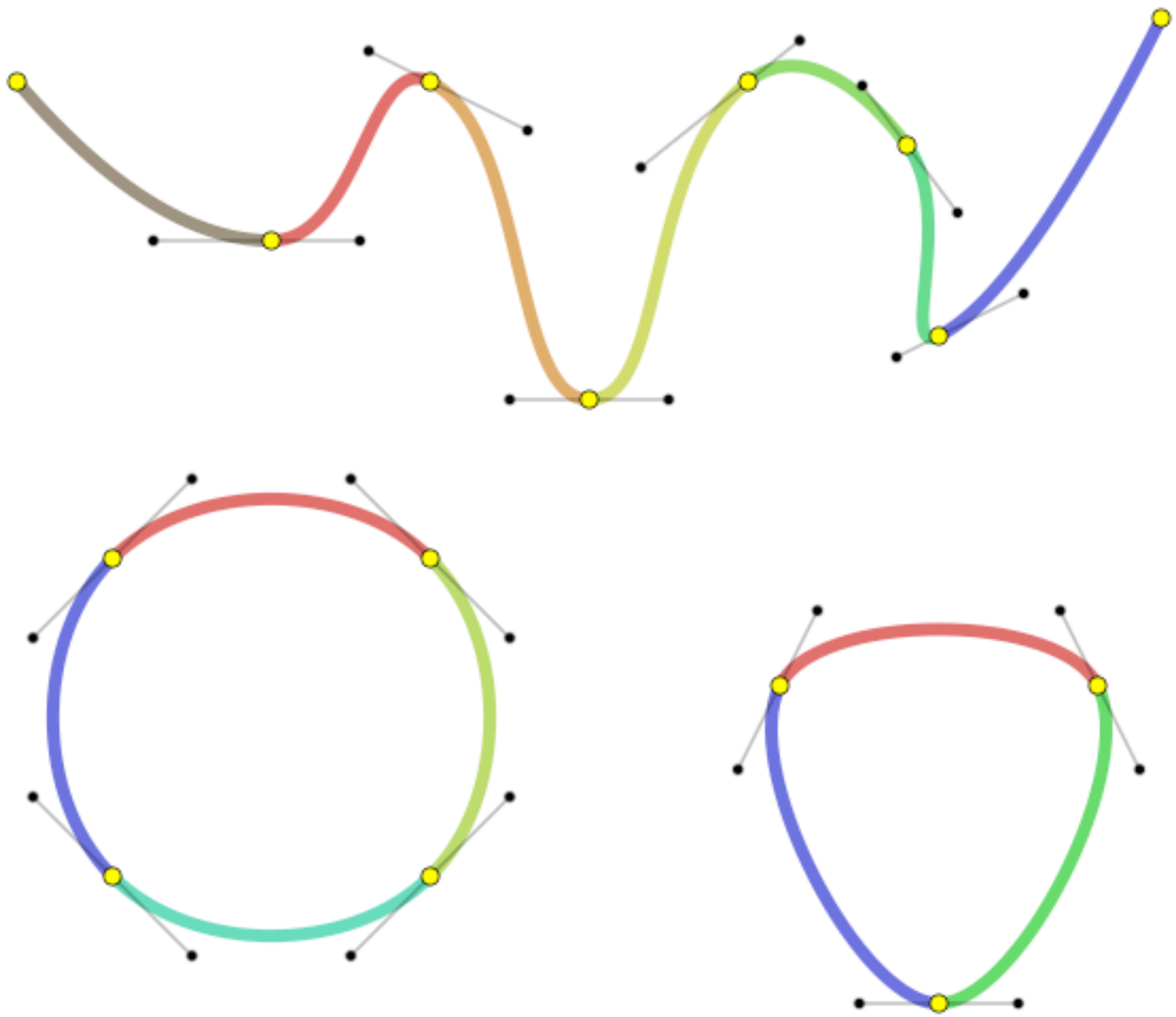
Interpolation: Spline



piecewise polynomial interpolation

Interpolation: Spline

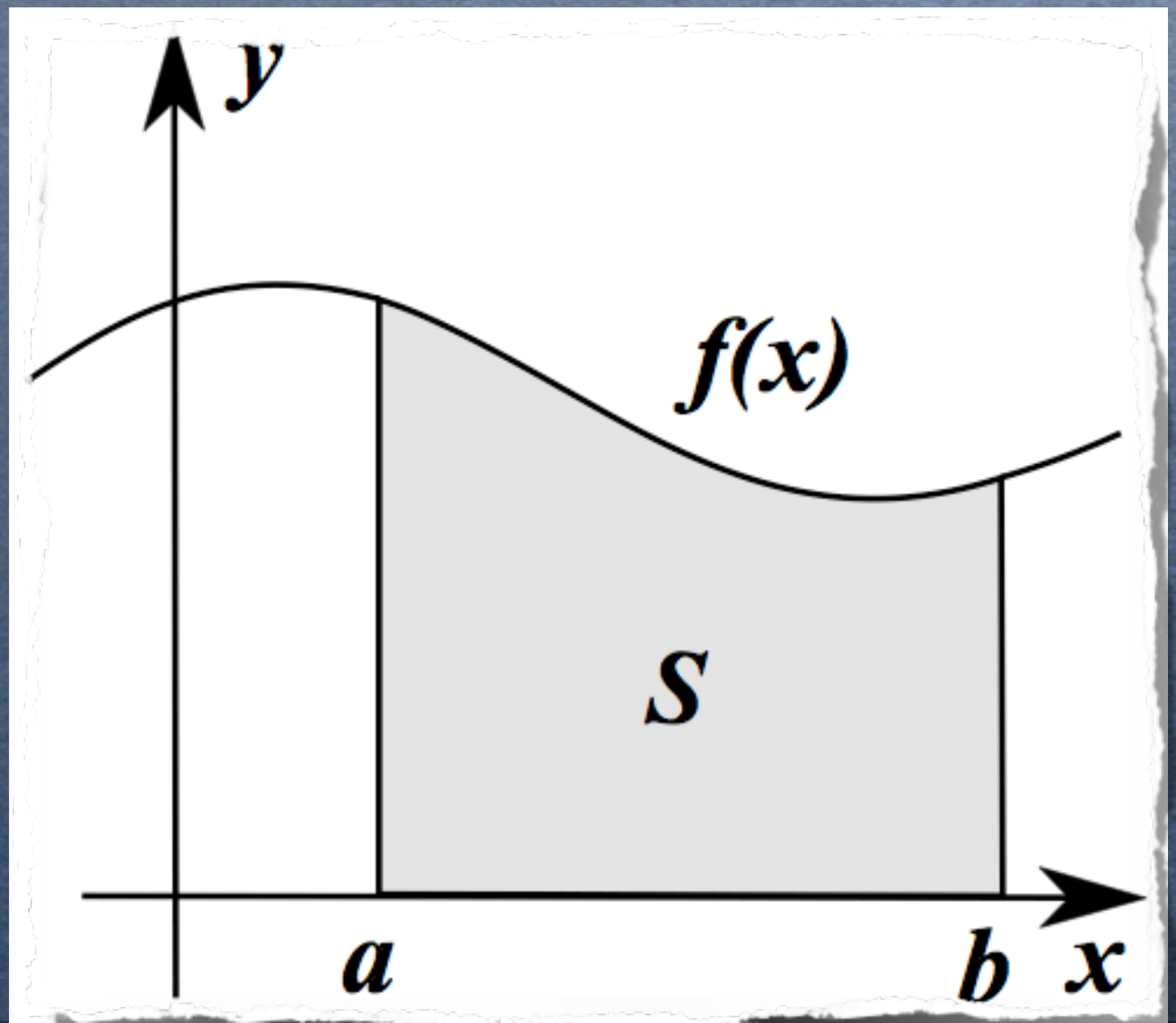




Interpolation: Spline

Integration:

$$\int_a^b f(x)\,\mathrm{d}x$$



Integration approximated by Summation

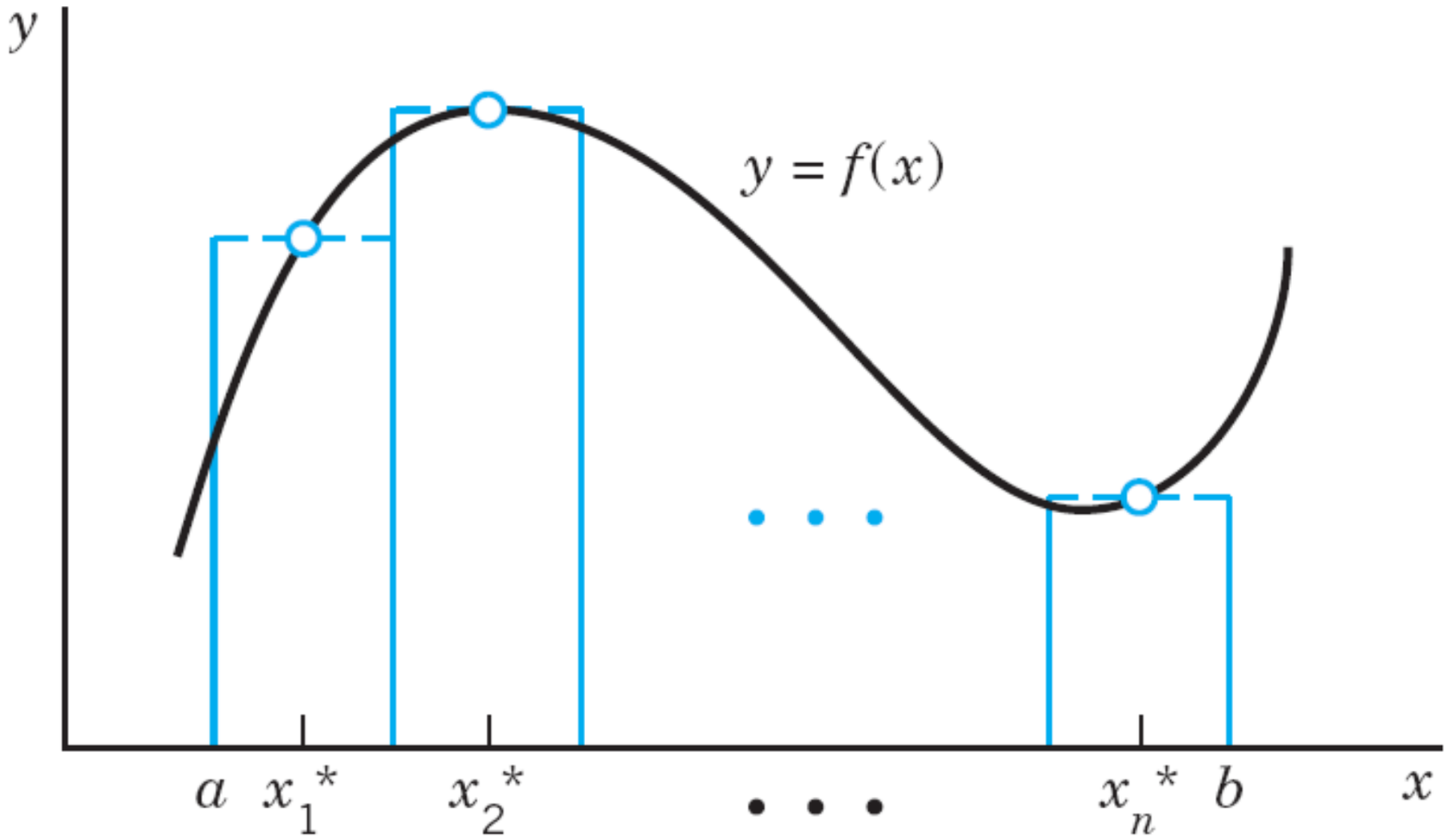
The numerical integration of a function $f(x)$ over some interval $[a, b]$ is a weighted sum of the function values at the sample points (nodes),

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{k=0}^{N} w_k\, f(x_k), \qquad \text{with} \quad a = x_0 < x_1 \cdots < x_N = b,$$

Integration : Middle-point rule

$$\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x \approx \Delta x f(\frac{x_k + x_{k+1}}{2}),$$

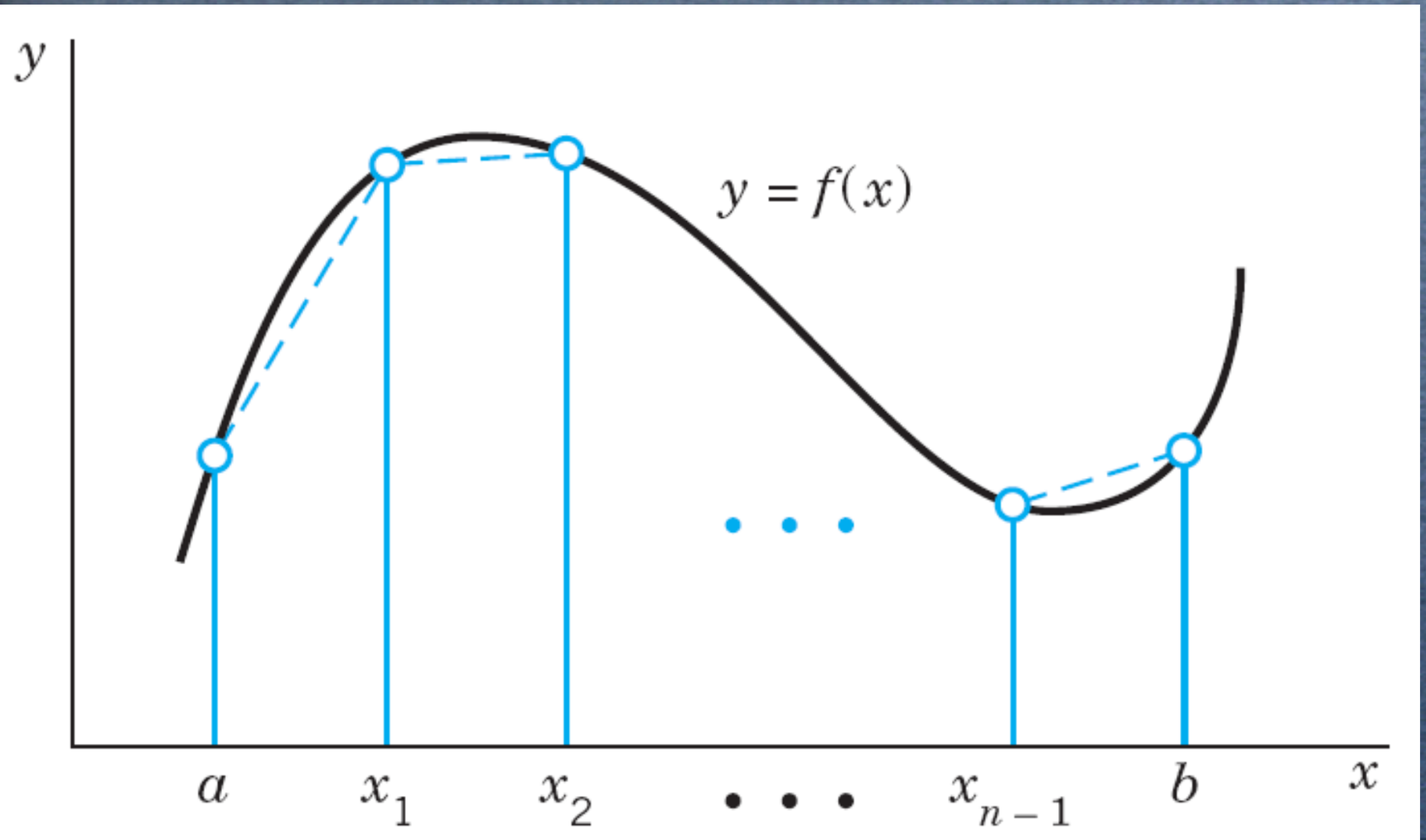
where $\Delta x = x_{k+1} - x_k$.



Integration: Trapezoidal rule

$$\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x \approx \frac{\Delta x}{2}[f(x_k) + f(x_{k+1})] + \mathbf{O}(\Delta x^3),$$

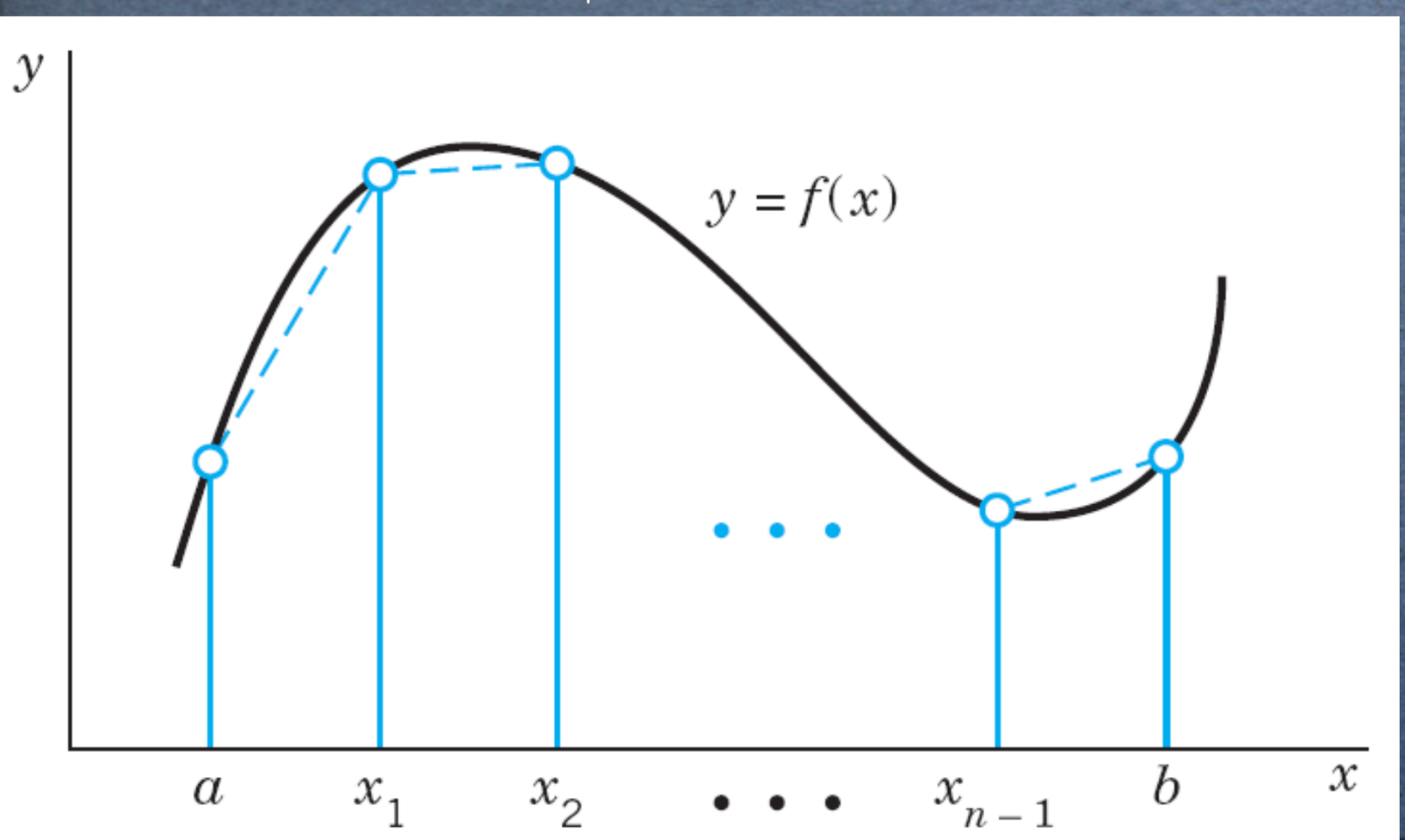
where $\Delta x = x_{k+1} - x_k$.



Integration: Trapezoidal rule

$$\int_b^a f(x)\,\mathrm{d}x \approx \Delta x\{\frac{[f(a)+f(b)]}{2}+\sum_{i=1}^{n-1} f(x_i)\},$$

where $\Delta x = x_{k+1} - x_k$.



Integration: Simpson's rule

Change the interval $x = \{x_k - \Delta x, x_k, x_k + \Delta_x\}$ to $t = \{-\Delta x, 0, \Delta x\}$ by $t = x - x_k$, then the second-degree polynomial to fit $f(t)$ is

$$P_2(t) = c_2 t^2 + c_1 t + c_0,$$

with the coefficients

$$\begin{aligned} c_0 &= f_k, \\ c_1 &= \frac{f_{k+1} - f_{k-1}}{2\Delta x}, \\ c_2 &= \frac{1}{2\Delta x^2}(f_{k+1} + f_{k-1} - 2f_k). \end{aligned}$$

Integrate the second-degree polynomial $f(x)$ from $t = -\Delta x$ to $t = \Delta x$,

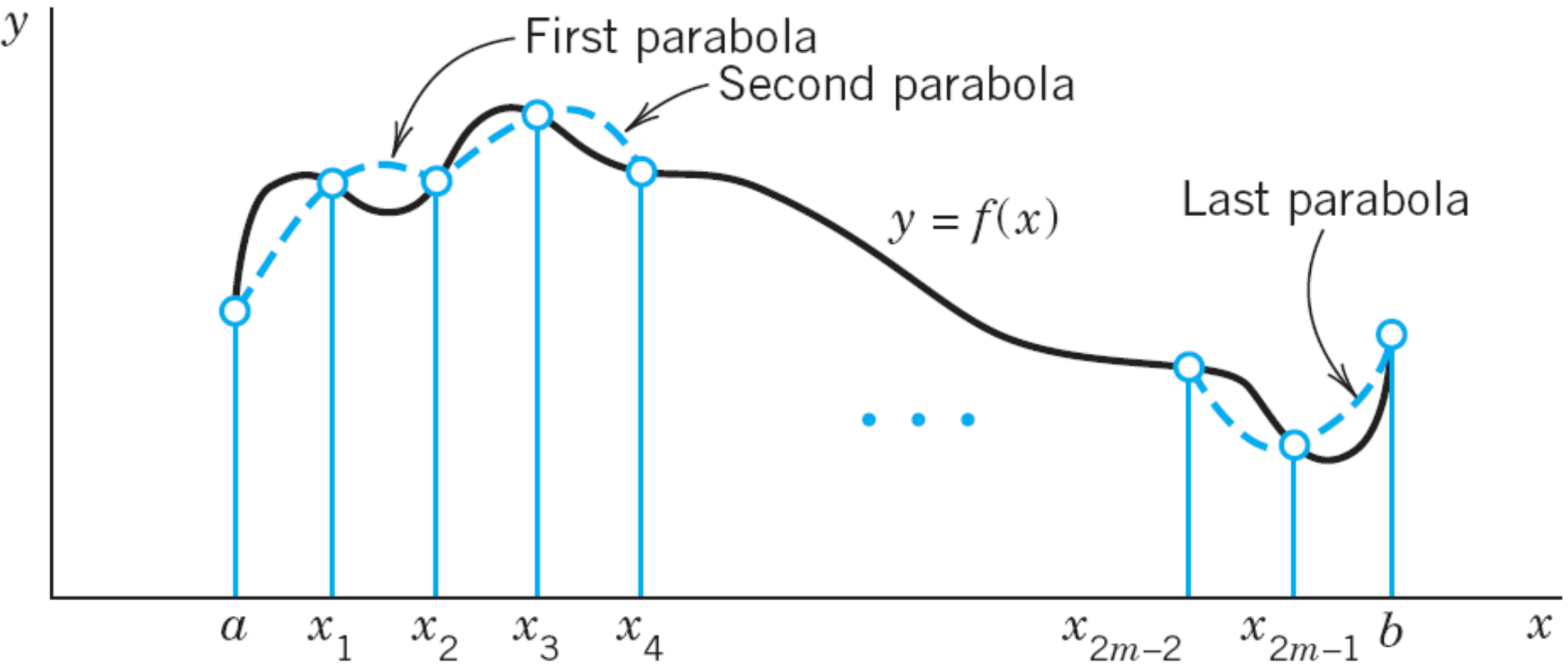
$$\int_{-\Delta x}^{\Delta x} P_2(t)\mathrm{d}t = \frac{\Delta x}{3}(f_{k-1} + 4f_k + f_{k+1}).$$

This is the Simpson integration formula.

Integration: Simpson's rule

$$\int_{x_{k-1}}^{x_{k+1}} f(x)\mathrm{d}x \approx \frac{\Delta x}{3}[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})] + \mathbf{O}(\Delta x^5),$$

where $\Delta x = x_{k+1} - x_k$.



Integration: Taylor expansion of integration

For

$$\begin{aligned} g(x) &= \int_{x_k}^{x} f(t)\mathrm{d}t, \\ &= g(x_k) + g'(x_k)\Delta x + \frac{1}{2!}g^{(2)}(x_k)\Delta x^2 + \frac{1}{3!}g^{(3)}(x_k)\Delta x^3 + \cdots \\ &= g(x_k) + f(x_k)\Delta x + \frac{1}{2!}f'(x_k)\Delta x^2 + \frac{1}{3!}f^{(2)}(x_k)\Delta x^3 + \cdots \end{aligned}$$

where $(x - x_k) = \Delta x$. Similarly

$$\begin{aligned} \int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x &= g(x_k) + f(x_k)\Delta x + \frac{1}{2!}f'(x_k)\Delta x^2 + \frac{1}{3!}f^{(2)}(x_k)\Delta x^3 + \cdots \\ \int_{x_k}^{x_{k-1}} f(x)\mathrm{d}x &= g(x_k) - f(x_k)\Delta x + \frac{1}{2!}f'(x_k)\Delta x^2 - \frac{1}{3!}f^{(2)}(x_k)\Delta x^3 + \cdots, \end{aligned}$$

where $g(x_k) = 0$. Then

$$\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x + \int_{x_{k-1}}^{x_k} f(x)\mathrm{d}x = 2[f(x_k)\Delta x + \frac{1}{3!}f^{(2)}(x_k)\Delta x^3 + \frac{1}{5!}f^{(4)}(x_k)\Delta x^5 + \cdots]$$

Integration: Errors

By replacing Δx by $\Delta x/2$, we have

$$\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x = f(x_k)\Delta x + \frac{1}{2^2 \cdot 3!} f^{(2)}(x_k)\Delta x^3 + \frac{1}{2^4 \cdot 5!} f^{(4)}(x_k)\Delta x^5 + \cdots]$$

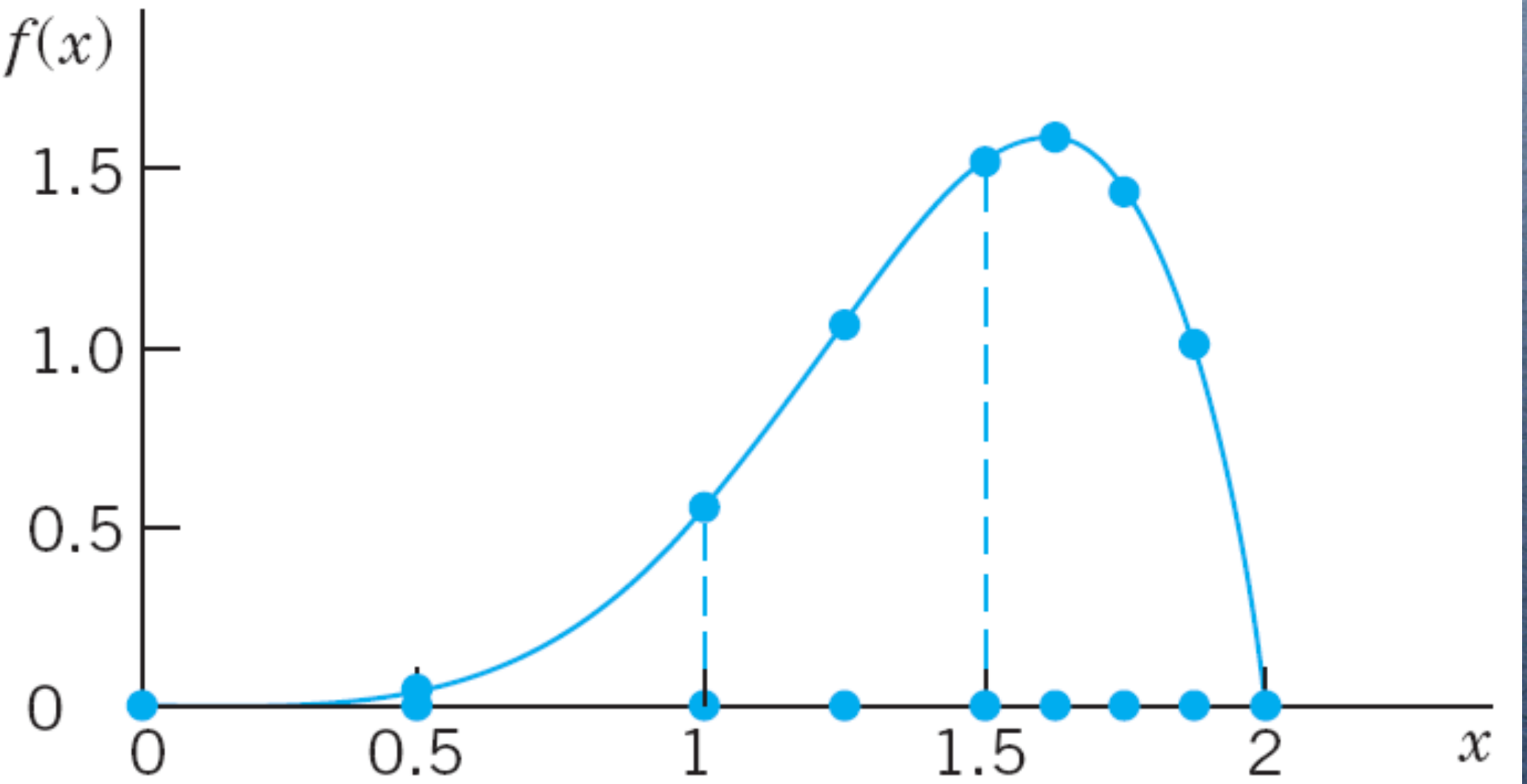
- The error of trapezoidal rule:

$$\int_{x_k}^{x_{k+1}} f(x)\mathrm{d}x - \frac{\Delta x}{2}[f(x_k) + f(x_{k+1})] = \mathbf{O}(\Delta x^3) + \ldots,$$

- The error of Simpson's rule:

$$\int_{x_{k-1}}^{x_{k+1}} f(x)\mathrm{d}x - \frac{\Delta x}{3}[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})] = \mathbf{O}(\Delta x^5) + \ldots$$

Integration: Adaptive method



Week 3: (3/20)

3. **Interpolation**: [T] Ch. 19.3, - 19.4; [C] Ch. 4
   - Lagrange polynomial
   - Runge phenomena
   - Chebyshev interpolation
   - Newton's divided difference interpolation
   - spline interpolation
   - Padé interpolation
4. **Numerical Integration**: [T] Ch. 19.5; [C] Ch. 5
   - Trapezoidal rule
   - Simpson's rule
   - Adaptive method
   - Gauss integration formula
5. **Numerical Linear Algebra**: [T] Ch. 20.1-20.3; [C] Ch. 7, 8
   - Gauss Elimination
   - LU-Factorization
   - Solutions by Iteration
   - Eigen-value problem

Matrix: Linear algebra

$$\bar{\bar{\mathbf{A}}} \cdot \vec{\mathbf{x}} = \vec{\mathbf{b}},$$

where

$$\bar{\bar{\mathbf{A}}} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ & \cdots & & \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix}, \vec{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \text{and} \quad \vec{\mathbf{b}} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}.$$

- $M = N$, nonsingular case,
- $M < N$, underdetermined case: minimum-norm solution,
- $M > N$, overdetermined case, Least-Squares-Error solution.

Gauss method: Forward Elimination

Consider $M = N = 3$,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix},$$

$$\begin{array}{rcrcrcl} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & = & b_2 \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & = & b_3 \end{array}$$

step 1: by pivoting at a_{11}

$$\begin{array}{rcrcrcl} a_{11}^{(0)}x_1 & + & a_{12}^{(0)}x_2 & + & a_{13}^{(0)}x_3 & = & b_1^{(0)} \\ & & a_{22}^{(1)}x_2 & + & a_{23}^{(1)}x_3 & = & b_2^{(1)} \\ & & a_{32}^{(1)}x_2 & + & a_{33}^{(1)}x_3 & = & b_3^{(1)} \end{array}$$

Gauss method: Forward Elimination

step 2: by pivoting at a_{22}

$$\begin{array}{ccccccc} a_{11}^{(0)}x_1 & + & a_{12}^{(0)}x_2 & + & a_{13}^{(0)}x_3 & = & b_1^{(0)} \\ & & a_{22}^{(1)}x_2 & + & a_{23}^{(1)}x_3 & = & b_2^{(1)} \\ & & & & a_{33}^{(2)}x_3 & = & b_3^{(2)} \end{array}$$

$$\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & 0 & a_{33}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(1)} \\ b_3^{(2)} \end{bmatrix},$$

- for subtraction $\frac{1}{3}N^3$ loops,
- for multiplication $\frac{1}{2}N^2M$ loops.

Gauss method: Backward Substitution

step 3: by backward substitution:

$$\begin{aligned} x_3 &= b_3^{(2)}/a_{33}^{(2)} \\ x_2 &= (b_2^{(1)} - a_{23}^{(1)} x_3)/a_2^{(1)} 2 \end{aligned}$$

$$x_1 = (b_1^{(0)} - \sum_{n=2}^{3} a_{1n}^{(0)} x_n)/a_{11}^{(0)}$$

in general form,

$$x_m = \left(b_m^{(m-1)} - \sum_{n=m+1}^{M} a_{mn}^{(m-1)} x_n \right) / a_{mm}^{(m-1)}, \quad \text{for} \quad m = M, M-1, \ldots, 1$$

- for backsubstituion $\frac{1}{2}N^2$ loops (one multiplication plus one subtraction)

Gauss method: Pivoting

If $a_{kk} = 0$, for example

$$\begin{bmatrix} 0 & 1 & 1 \\ 2 & -1 & -1 \\ 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(0)} \\ b_2^{(1)} \\ b_3^{(2)} \end{bmatrix},$$

- apply *row switching* to avoid the zero division,
- interchange rows only: *partial pivoting*,
- interchange row and columns: *full pivoting*,
- simply pick the largest (in magnitude) element as the pivot,
- more useful to reduce the round-off error,

$$\text{Max}\{|a_{mk}|, k \le m \le M\}$$

- a better choice,

$$\text{Max}\{\frac{|a_{mk}|}{\text{Max}\{|a_{mn}, k \le n \le M\}}, k \le m \le M\},$$

Matrix: LU-Decomposition (Factorization)

$$\bar{\bar{\mathbf{L}}} \cdot \bar{\bar{\mathbf{U}}} = \bar{\bar{\mathbf{A}}},$$

where $\bar{\bar{\mathbf{L}}}$ is *lower* triangular and $\bar{\bar{\mathbf{U}}}$ is upper triangular,

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

Matrix: LU-Decomposition (Factorization)

We can use a decomposition to solve the linear set

$$\bar{\bar{\mathbf{A}}} \cdot \vec{\mathbf{x}} = (\bar{\bar{\mathbf{L}}} \cdot \bar{\bar{\mathbf{U}}}) \cdot \vec{\mathbf{x}} = \bar{\bar{\mathbf{L}}} \cdot (\bar{\bar{\mathbf{U}}} \cdot \vec{\mathbf{x}}) = \vec{\mathbf{b}}$$

by first solving for the vector $\vec{\mathbf{y}}$

$$\boxed{\bar{\bar{\mathbf{L}}} \cdot \vec{\mathbf{y}} = \vec{\mathbf{b}},}$$

then solving

$$\boxed{\bar{\bar{\mathbf{U}}} \cdot \vec{\mathbf{x}} = \vec{\mathbf{y}}.}$$

Matrix: LU dicomposition

- forward substitution:

$$y_1 = \frac{b_1}{\alpha_{11}}$$

$$y_i = \frac{1}{\alpha_{ii}}[b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j], \quad i = 2, 3, \ldots, N$$

- backward substitution:

$$x_N = \frac{y_N}{\beta_{NN}}$$

$$x_i = \frac{1}{\beta_{ii}}[y_i - \sum_{j=1+1}^{N} \beta_{ij} x_j], \quad i = N-1, N-2, \ldots$$

LU decomposition, Crout's algorithm

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix},$$

with

$$\alpha_{i1}\beta_{1j} + \cdots + \alpha_{ii}\beta_{jj} = a_{ij}$$

there are N^2 equations for the $N^2 + N$ unknown α's and β's.

- set $\alpha_{ii} \equiv 1$, for $i = 1, \ldots, N$,
- for $i = 1, 2, \ldots, j$, solve β_{ij}

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}$$

- for $ji = j + 1, j + 2, \ldots, N$, solve α_{ij}

$$\alpha_{ij} = \frac{1}{\beta_{jj}}(\alpha_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta kj)$$

First-order ODEs: Order

- If the nth derivative $y^{(n)} = \mathrm{d}^n y/\mathrm{d}\, x^n$ of the unknown function $y(x)$ is the highest occurring derivative, it is called an ODE of nth-order:

$$F(x, y, y', \cdots, y^{(n)}) = 0, \qquad \text{where} \quad y^{(n)} = \frac{\mathrm{d}^n y}{\mathrm{d}\, x^n},$$

- Linear nth-order ODE:

$$y^{(n)} + p_{n-1}(x)y^{(n-1)} + \cdots + p_1(x)y' + p_0(x)y = r(x),$$

- Explicit form:

$$y' = f(x, y)$$

- Implicit form:

$$F(x, y, y') = 0$$

ODE: Euler's method

For a first-order differential equation:

$$\boxed{y'(t) + a\,y(t) = r, \quad \text{with} \quad y(0) = y_0,}$$

has the following analytical solution,

$$y(t) = (y_0 - \frac{r}{a})e^{-at} + \frac{r}{a}.$$

Euler's method:

$$\frac{y(t+h) - y(t)}{h} + a\,y(t) = r,$$

$$y(t+h) = [1 - ah]y(t) + hr, \quad \text{with} \quad y(0) = y_0.$$

with error of $\mathbf{O}(h)$.

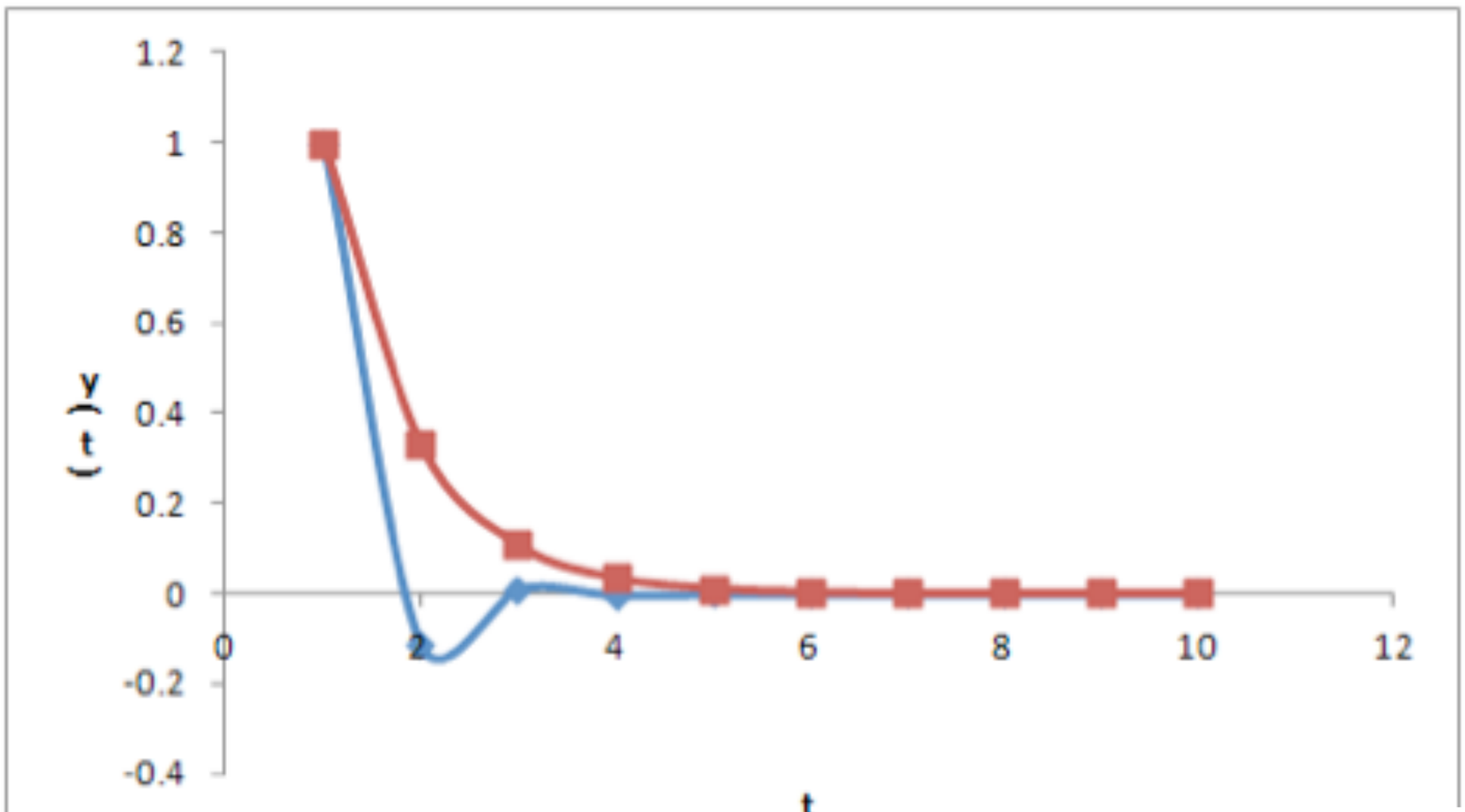
Homework 3: Euler's method

Solve the first-order differential equation:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -y.$$

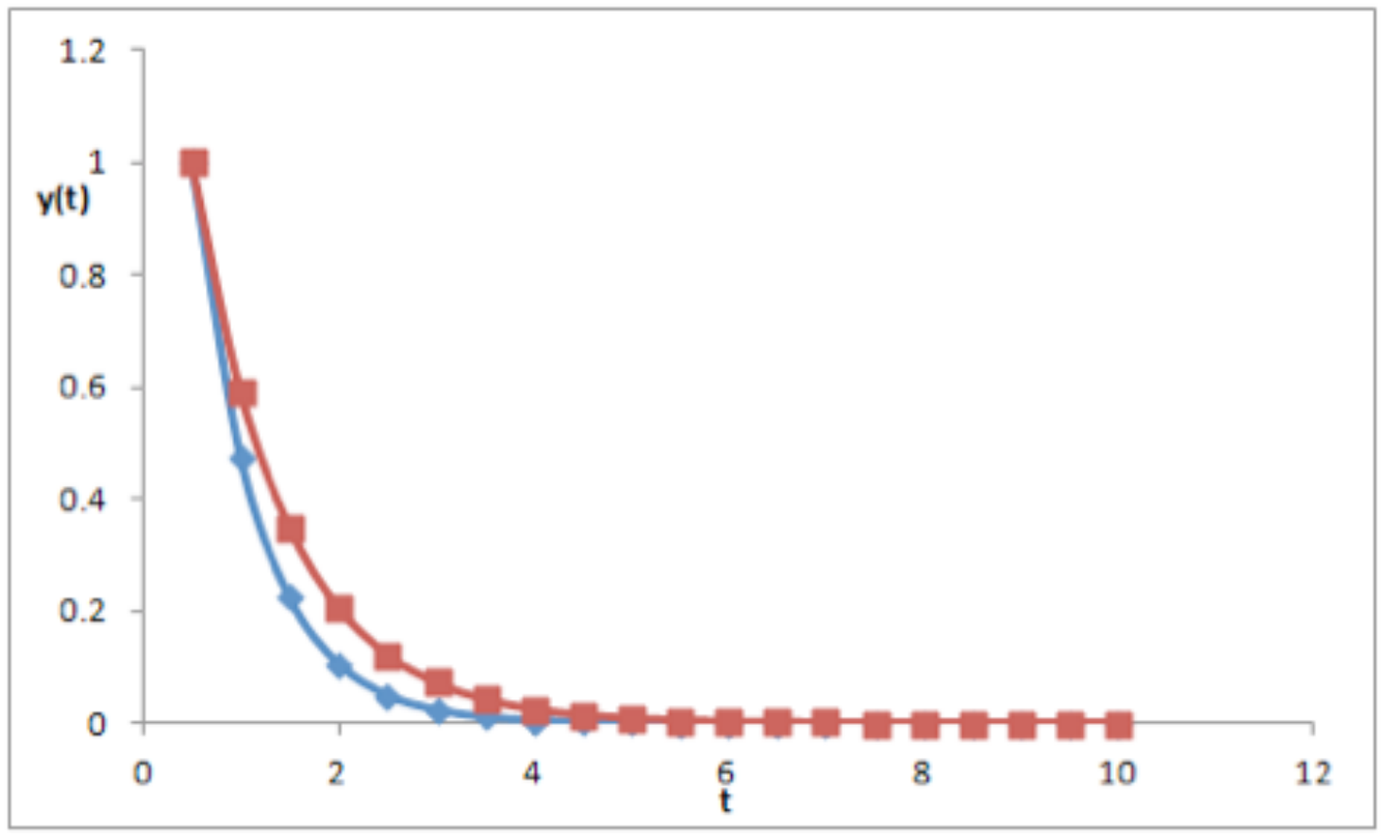
- For the initial condition $y(t=0) = 1$, plot $y(t)$ by using *Euler's method* for $t = [0, 10]$.
- Compare your results with the analytical solution,

$$y(t) = e^{-t}.$$

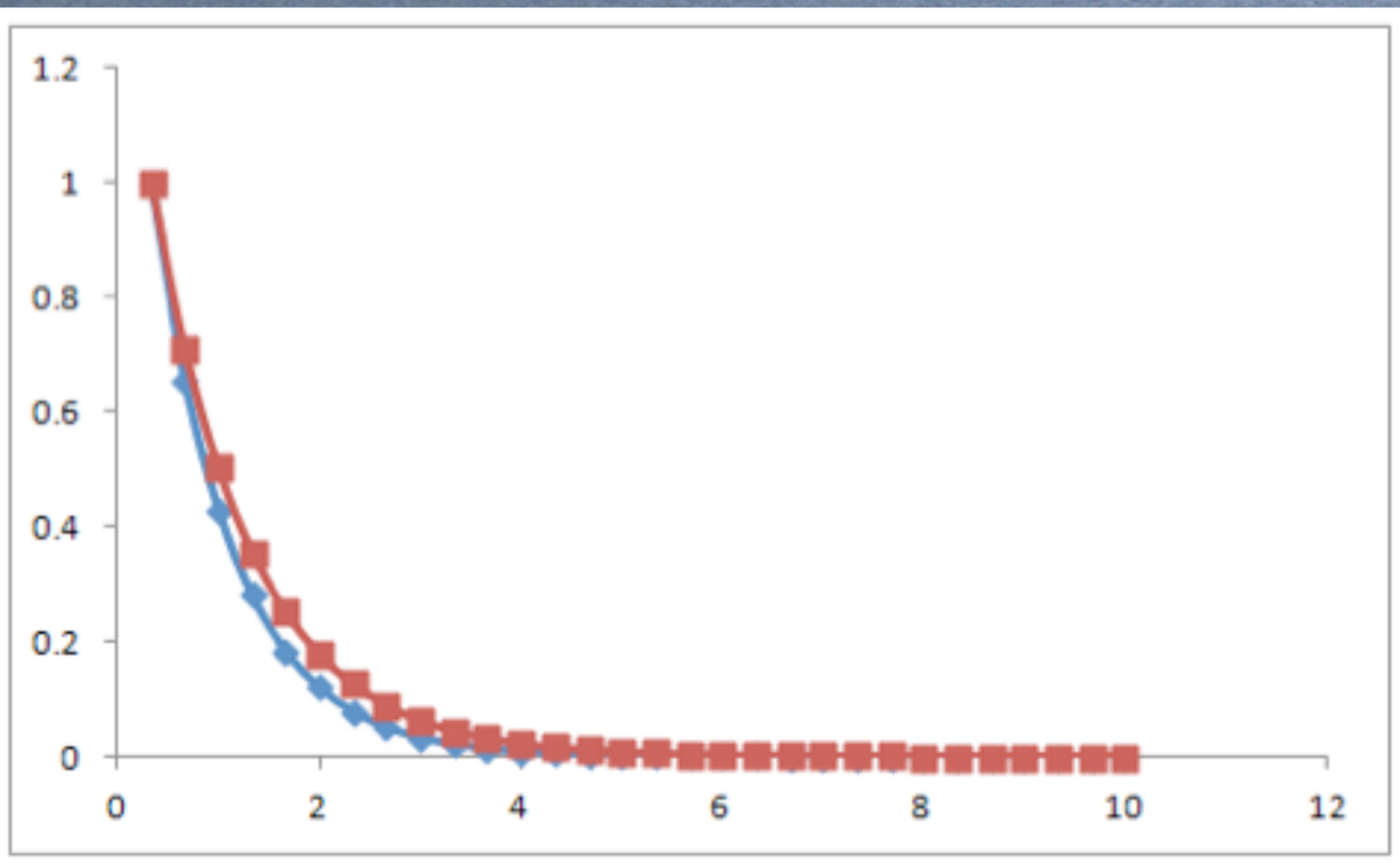

1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
y (t)
0
2
4
6
8
10
12
t

N = 10

N = 20


1.2
1
0.8
0.6
0.4
0.2
0
y(t)
0
2
4
6
8
10
12
t

Homework 3: Euler's method



N = 30

Homework 3: Euler's method

Solve the spring-mass equation (2nd-order ODE):

$$m\,\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx,$$

by defining a coupled set of 1st-order ODEs

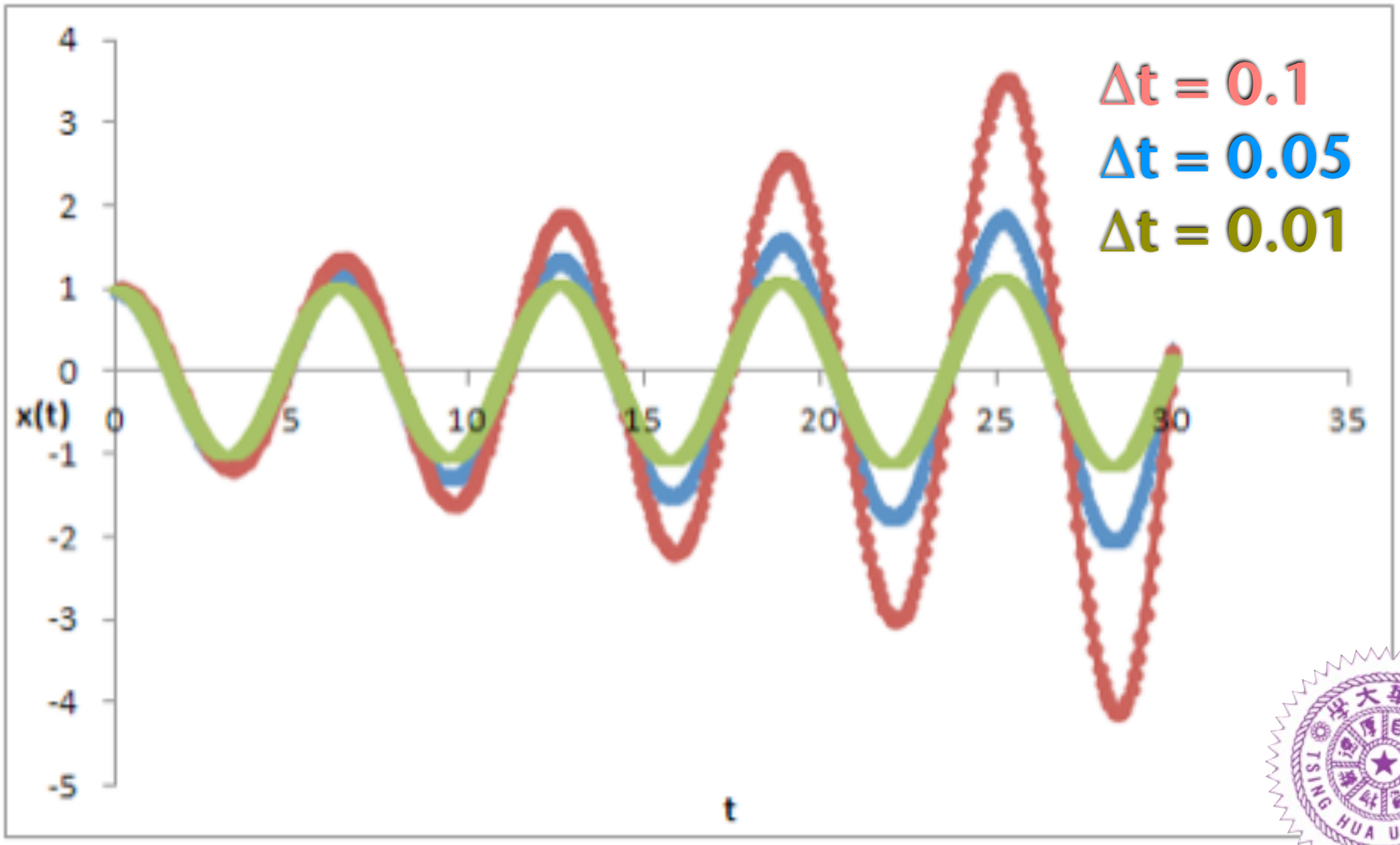
$$\begin{aligned} a \equiv \frac{\mathrm{d}v}{\mathrm{d}t} &= -\frac{k}{m}x, \\ \frac{\mathrm{d}x}{\mathrm{d}t} &= v, \end{aligned}$$

with $k/m = 1$.

- ☐ For the initial conditions $x(t=0) = 1$ and $v(t=0) = 0$, plot $x(t)$ by using *Euler's method* for $t = [0, 30]$.
- ☐ Compare your results with the analytical solution

$$x(t) = \mathrm{Cos}\, t.$$

Homework 3: Euler's method



Week 3: (3/27)

6. Numerical ODEs: [T] Ch. 21.1; [C] Ch. 10

- **Euler's method**
- **Heun's method**
- **Runge-Kutta method**
- **Runge-Kutta-Fehlberg method**
- **Stiff ODEs**
- **Backward Euler method**
- **Adams-Bashforth method**
- **Richard's method**

ODE: Euler's method

For a first-order differential equation:

$$y'(t) + a\,y(t) = r, \quad \text{with} \quad y(0) = y_0,$$

has the following analytical solution,

$$y(t) = (y_0 - \frac{r}{a})e^{-at} + \frac{r}{a}.$$

Euler's method:

$$\frac{y(t+h) - y(t)}{h} + a\,y(t) = r,$$

$$y(t+h) = [1 - ah]y(t) + hr, \quad \text{with} \quad y(0) = y_0.$$

with error of $\mathbf{O}(h)$.

ODE: Heun's trapezoidal method

For a first-order differential equation:

$$y'(t) = f(t, y),$$

$$y(t)|_{t_k}^{t_{k+1}} = y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} f(t, y)\mathrm{d}t,$$

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y)\mathrm{d}t, \quad \text{with} \quad y(t_0) = y_0.$$

By trapezoidal integration method:

$$y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k), f(t_{k+1}, y_{k+1})]$$

where y_{k+1} is unknown and can be replaced by $y_{k+1} \approx y_k + hf(t_k, y_k)$.
Heun's method:

$$\boxed{y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k), f(t_{k+1}, y_k + hf(t_k, y_k))]}$$

with error of $\mathbf{O}(h^2)$.

Finite Difference Approximation

$$\frac{\mathrm{d}}{\mathrm{d}x}y(x)|_{x=x_j} \approx \frac{y(x_j)-y(x_{j-1})}{x_j-x_{j-1}}$$

- Taylor's expansion:

$$u(x_{j+1}) = u(x_j)+u'(x_j)\Delta x+\frac{u''(x_j)}{2}(\Delta x)^2+\frac{u'''(x_j)}{3!}(\Delta x)^3+\ldots,$$

$$u(x_{j-1}) = u(x_j)-u'(x_j)\Delta x+\frac{u''(x_j)}{2}(\Delta x)^2-\frac{u'''(x_j)}{3!}(\Delta x)^3+\ldots,$$

- Euler's 2nd-order FD approximation:

$$\begin{aligned} u'(x_j) &= \frac{u(x_{j+1})-u(x_{j-1})}{2\Delta x}-\frac{u'''(x_j)}{2*3!}(\Delta x)^2+\ldots, \\ &\approx \frac{u(x_{j+1})-u(x_{j-1})}{2\Delta x}+\mathbf{O}(\Delta x^2), \end{aligned}$$

- 4th-order FD method:
- Runge-Kutta method:
- Differential matrix:

ODE: Runge-Kutta method, 2nd-order

Second-order Runge-Kutta method:

$$k_1 = hf(t_k, y_k),$$

$$k_2 = hf(t_k + \frac{1}{2}h, y_k + \frac{1}{2}k_1),$$

$$y_{k+1} = y_k + k_2 + \mathbf{O}(h^3).$$

Generalized: Runge-Kutta method, 2nd-order

$$
\begin{aligned}
y_{k+1} &= y_k + c_1\, k_1 + c_2\, k_2 + \mathbf{O}(h^3), \\
k_1 &= h f(t_k, y_k), \\
k_2 &= h f(t_k + a_1\, h, y_k + b_1\, k_1).
\end{aligned}
$$

Proof: Runge-Kutta method, 2nd-order

$$
\begin{aligned}
y_{k+1} &= y_k + c_1\, k_1 + c_2\, k_2 + \mathbf{O}(h^3), \\
&= y(t_k) + h\, y'(t_k, y_k) + \frac{h^2}{2!} y''(t_k, y_k) + \mathbf{O}(h^3), \\
&= y(t_k) + h\, f(t_k, y_k) + \frac{h^2}{2!} f'(t_k, y_k) + \mathbf{O}(h^3), \\
&= y(t_k) + h\, f(t_k, y_k) + \frac{h^2}{2!} \left[\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{\mathrm{d}y}{\mathrm{d}x} \right]_{(t_k, y_k)} + \mathbf{O}(h^3), \\
&= y(t_k) + h\, f(t_k, y_k) + \frac{h^2}{2!} \left[\frac{\partial f}{\partial t} + f(t_k, y_k) \frac{\partial f}{\partial y} \right]_{(t_k, y_k)} + \mathbf{O}(h^3),
\end{aligned}
$$

Proof: Runge-Kutta method, 2nd-order

$$
\begin{aligned}
y_{k+1} &= y(t_k) + h\, f(t_k, y_k) + \frac{h^2}{2!}\left[\frac{\partial f}{\partial t} + f(t_k, y_k)\,\frac{\partial f}{\partial y}\right]_{(t_k, y_k)} + \mathbf{O}(h^3), \\
&= y_k + c_1\, k_1 + c_2\, k_2 + \mathbf{O}(h^3), \\
&= y(t_k) + c_1\, h\, f(t_k, y_k) + c_2\, h\, f(t_k + a_1\, h, y_k + b_1\, h) + \mathbf{O}(h^3), \\
&= y(t_k) + c_1\, h\, f(t_k, y_k) + c_2\, h\left[f(t_k, y_k) + a_1\, h\,\frac{\partial f}{\partial t} + b_1\, h\,\frac{\partial f}{\partial y}\right]_{(t_k, y_k)} + \mathbf{O}(h^3),
\end{aligned}
$$

Generalized: Runge-Kutta method, 2nd-order

$$\begin{aligned} c_1 + c_2 &= 1, \\ a_1\, c2 &= \frac{1}{2}, \\ b_1\, c2 &= \frac{1}{2}. \end{aligned}$$

- Heun's method:

$$\begin{aligned} c_2 = \frac{1}{2}, c_1 = \frac{1}{2}, a_1 = 1, b_1 = 1, \\ y_{k+1} = y_k + (\frac{1}{2}k_1 + \frac{1}{2}k_2), \\ k_1 = hf(t_k, y_k), \\ k_2 = hf(t_k + h, y_k + k_1). \end{aligned}$$

Generalized: Runge-Kutta method, 2nd-order

- Midpoint's method:

$$c_2 = 1, c_1 = 0, a_1 = \frac{1}{2}, b_1 = \frac{1}{2},$$

$$y_{k+1} = y_k + k_2,$$

$$k_1 = hf(t_k, y_k),$$

$$k_2 = hf(t_k + \frac{1}{2}\,h, y_k + \frac{1}{2}\,k_1).$$

- Ralston's method:

$$c_2 = \frac{2}{3}, c_1 = \frac{1}{3}, a_1 = \frac{3}{4}, b_1 = \frac{3}{4},$$

$$y_{k+1} = y_k + (\frac{1}{3}k_1 + \frac{2}{3}k_2),$$

$$k_1 = hf(t_k, y_k),$$

$$k_2 = hf(t_k + \frac{3}{4}\,h, y_k + \frac{3}{4}\,k_1).$$

ODE: Runge-Kutta method, 4th-order

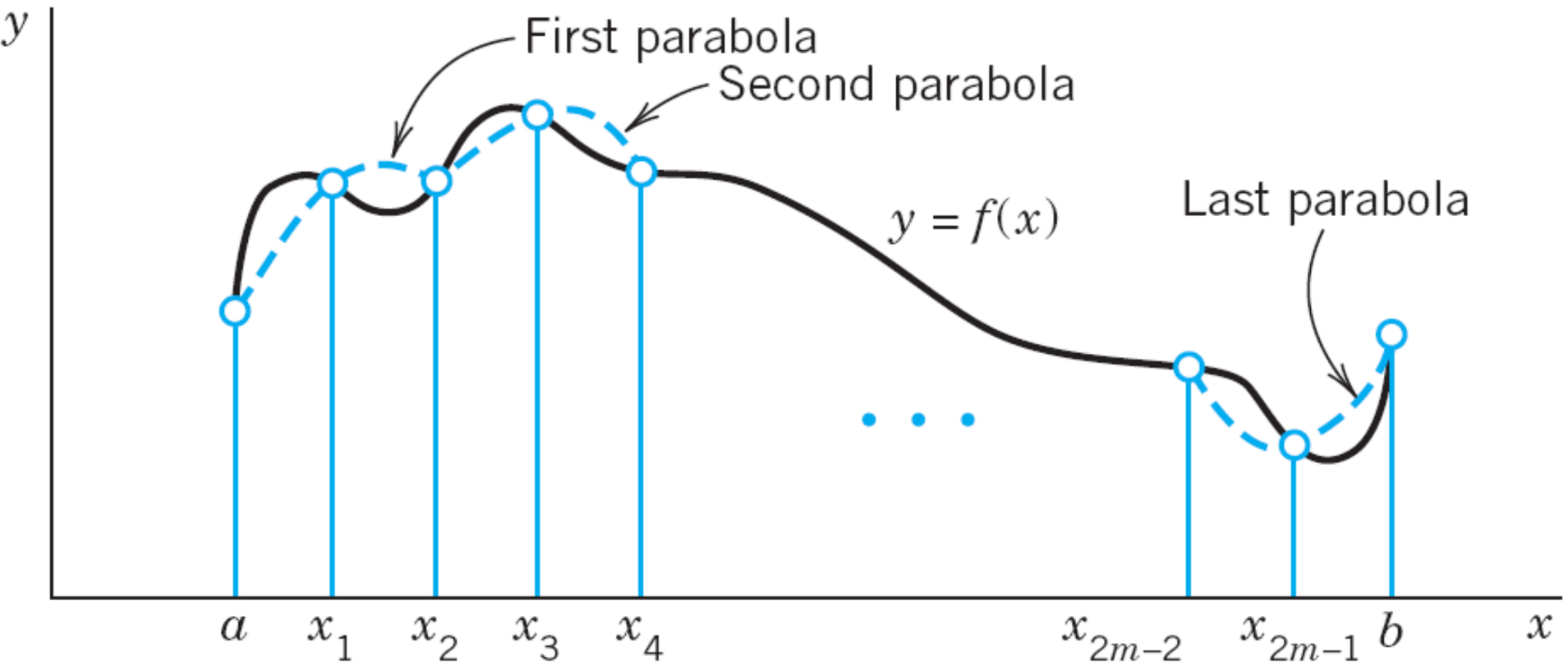
Fourth-order Runge-Kutta method:

$$
\begin{aligned}
k_1 &= hf(t_k, y_k), \\
k_2 &= hf(t_k + \frac{1}{2}h, y_k + \frac{1}{2}k_1), \\
k_3 &= hf(t_k + \frac{1}{2}h, y_k + \frac{1}{2}k_2), \\
k_2 &= hf(t_k + h, y_k + k_3), \\
y_{k+1} &= y_k + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathbf{O}(h^5).
\end{aligned}
$$

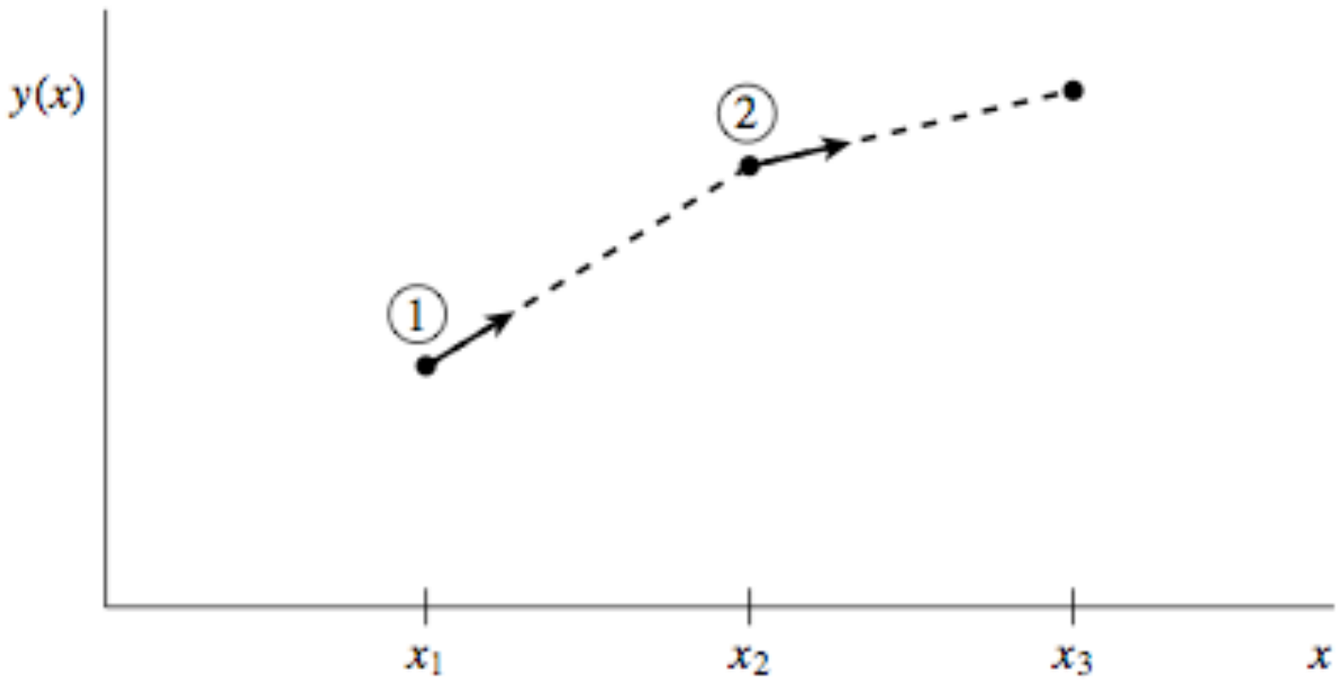
Integration: Simpson's rule

$$\int_{x_{k-1}}^{x_{k+1}} f(x)\mathrm{d}x \approx \frac{\Delta x}{3}[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})] + \mathbf{O}(\Delta x^5),$$

where $\Delta x = x_{k+1} - x_k$.

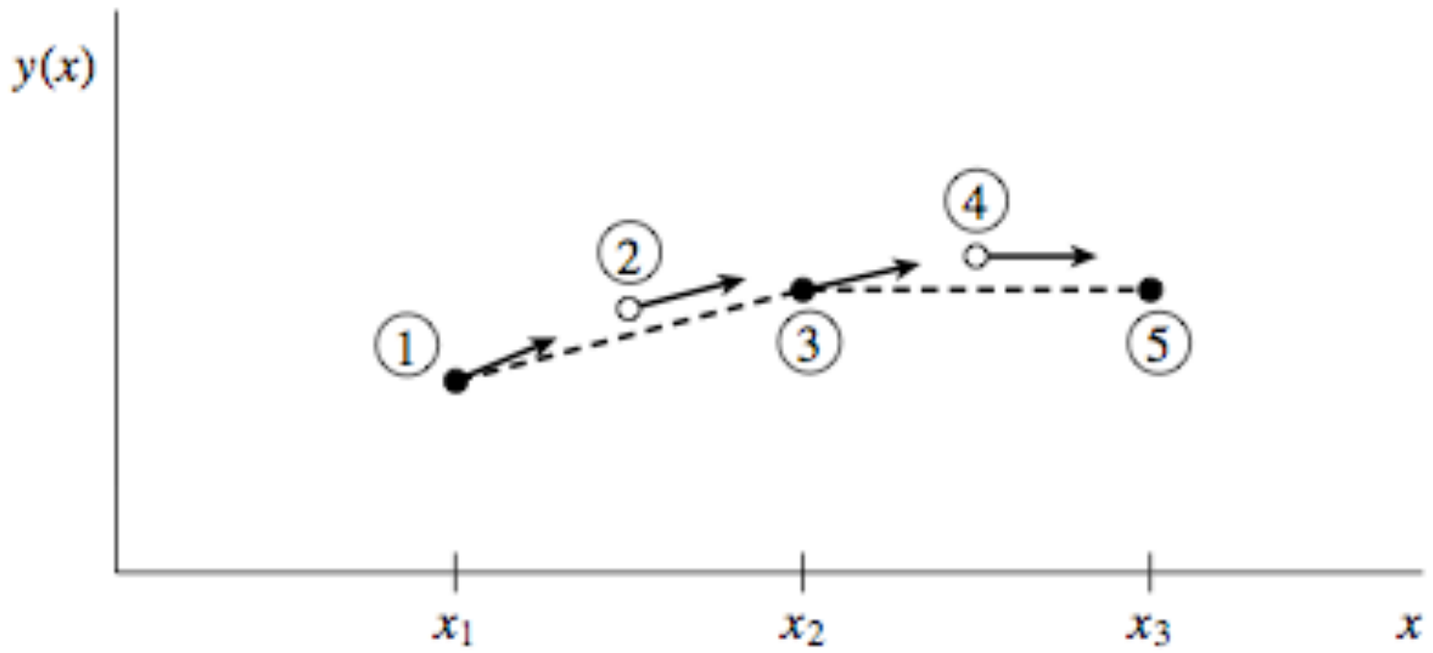


ODE: Runge-Kutta method,

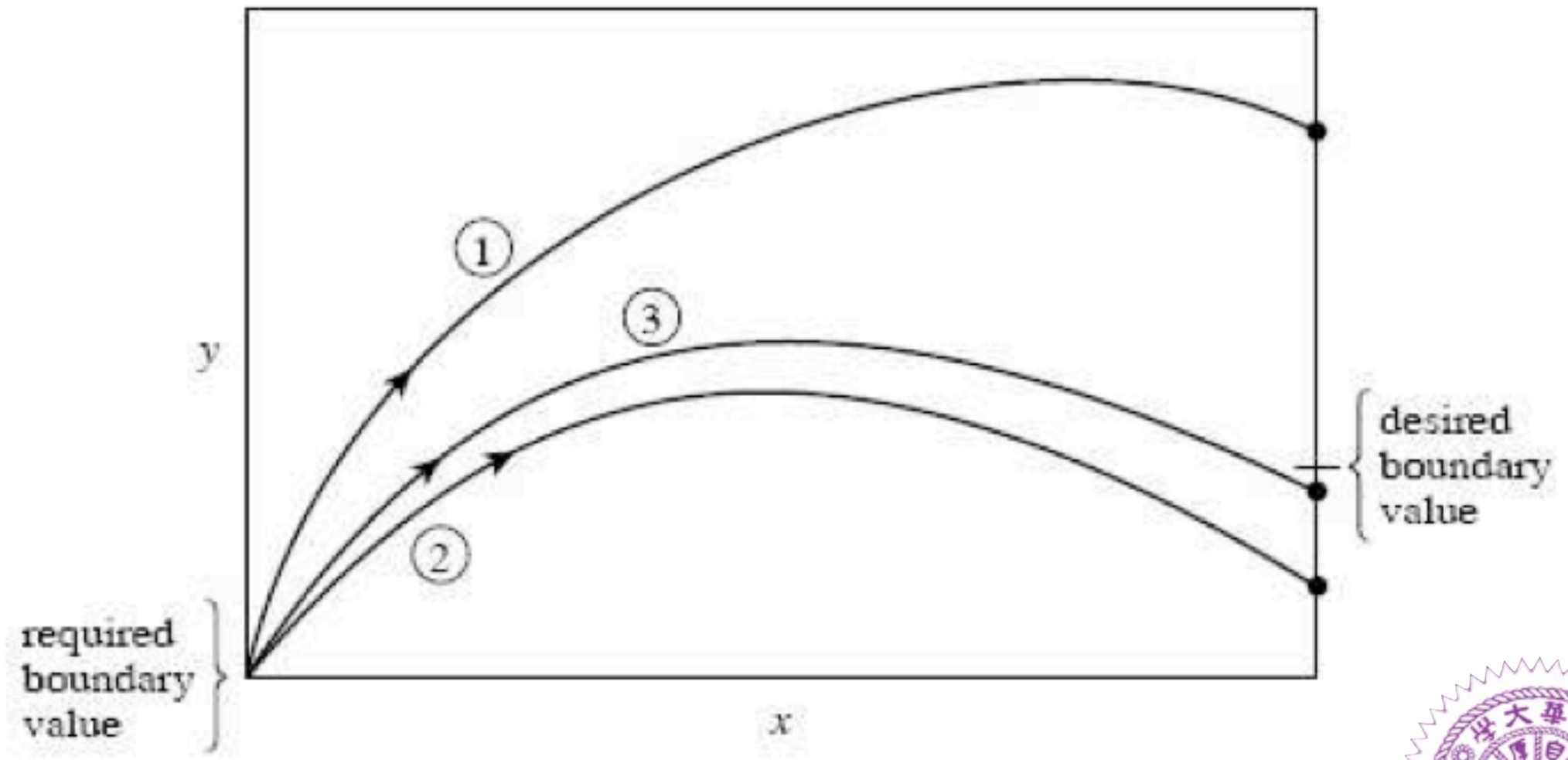


2nd-order

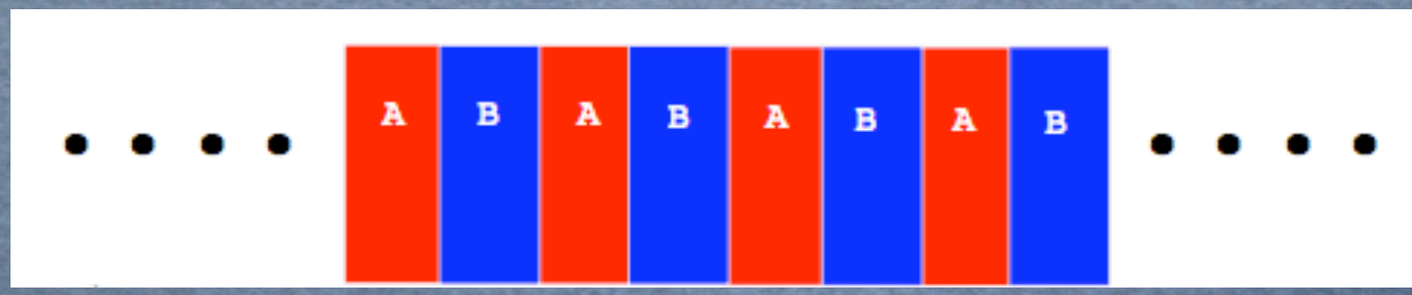
4th-order



ODE: Boundary-Value Problem



Homework 4: Coupled-mode equation



coupled-mode equation:

$$\frac{dE_+(z)}{dz} = i\delta E_+(z) + i\kappa E_-(z)$$

$$\frac{dE_-(z)}{dz} = -i\delta E_-(z) - i\kappa^* E_+(z)$$

with the Boundary Condition:

$$E_+(z=0) = 1$$

$$E_-(z=L) = 0$$

Homework 4: Coupled-mode equation

- Coupling coefficient:

$$\kappa(z) = \kappa_0 \times \mathbf{F}(z),$$

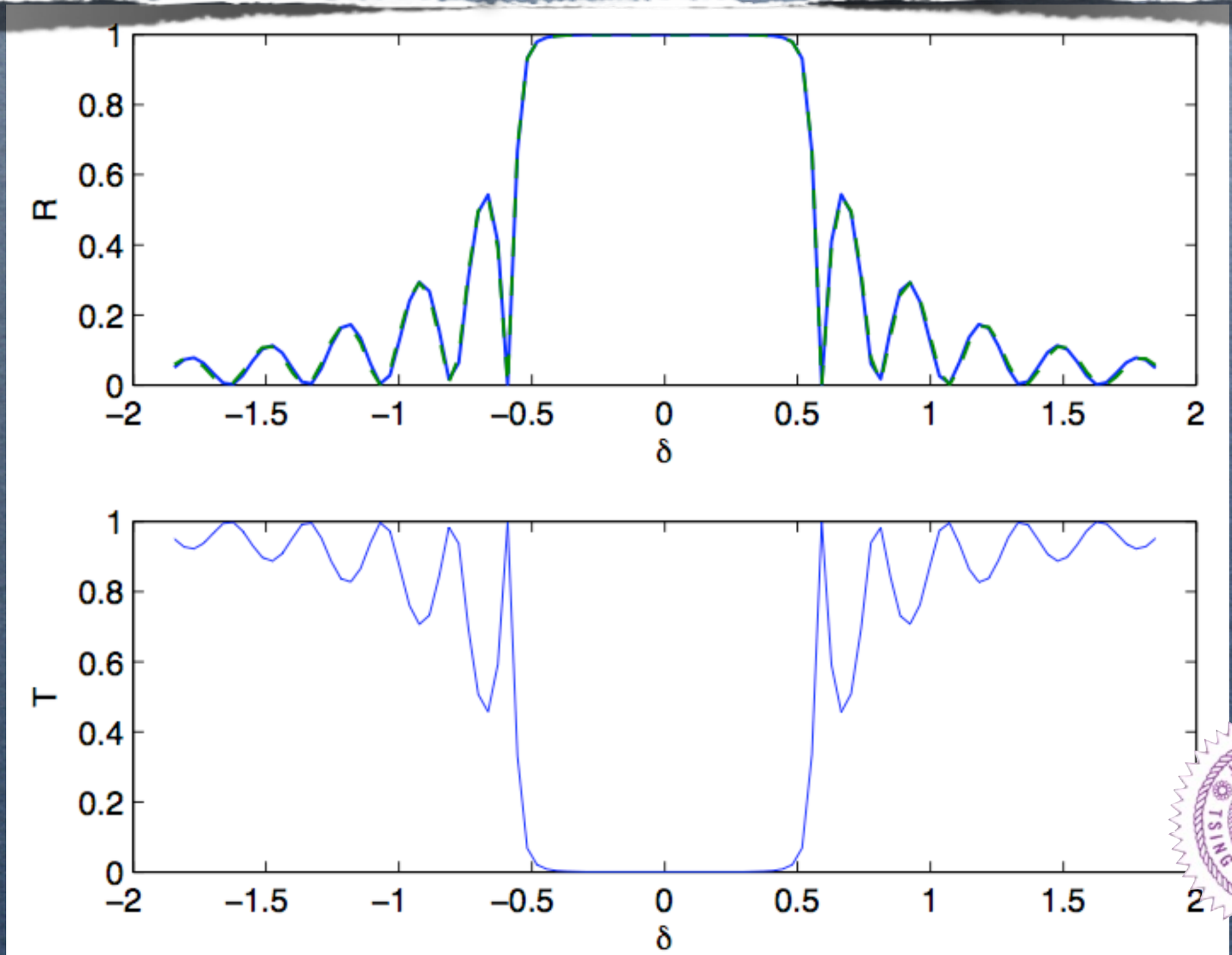
- Assume $\kappa_0 = 0.5$, and $\mathbf{F}(z) = 1$, i.e., κ is a constant: calculate

1. Tranmission spectrum: $T \equiv |\frac{E_+(z=L)}{E_+(z=0)}|^2$ v.s. detuning δ;
2. Reflection spectrum: $R \equiv |\frac{E_-(z=0)}{E_+(z=0)}|^2$ v.s. detuning δ.
3. Compare to the analytical solution,

$$R = \frac{\sinh^2 \sqrt{(\kappa L)^2 - (\delta L)^2}}{-\frac{\delta^2}{\kappa^2} + \cosh^2 \sqrt{(\kappa L)^2 - (\delta L)^2}},$$

where $L = 10$[mm] is the grating length, and estimate the absolute error in your simulation.

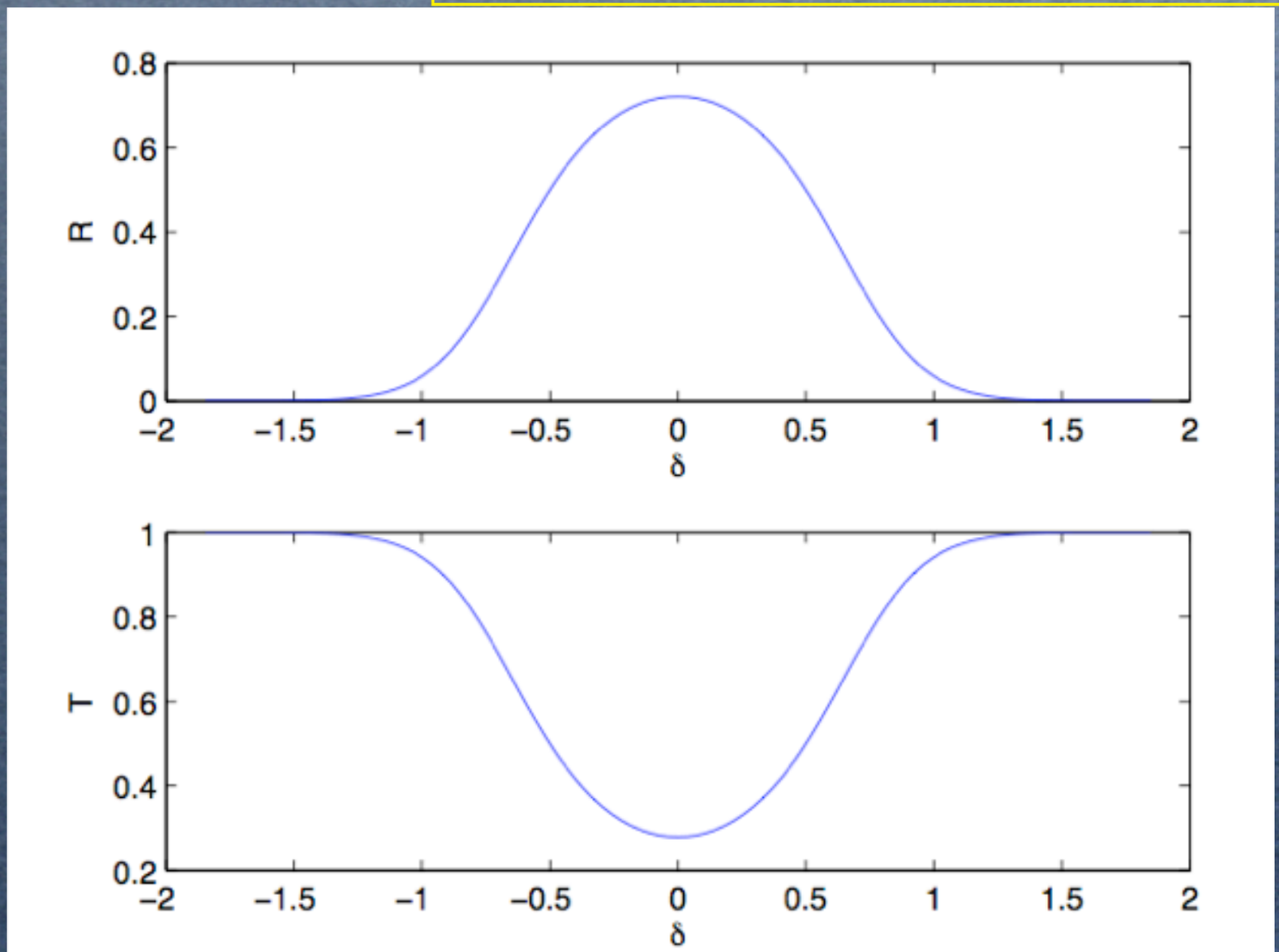
Homework 4: Coupled-mode equation



Homework 4: Coupled-mode equation

Apodization:

$$\mathbf{F}(z) = exp[-0.5 * (z - L/2)^2].$$



Reduce a nth-order ODE into a system

- An nth-order ODE:

$$y^{(n)} = F(t, y, y', \cdots, y^{(n-1)}),$$

can be converted to a system of n first-order ODEs by setting

$$y_1 = y, \quad y_2 = y', \quad y_3 = y'', \cdots, y_n = y^{(n-1)}.$$

- This system for an nth-order ODE is

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_3, \\ &\cdots \\ y_n' &= F(t, y_1, y_2, \cdots, y_n). \end{aligned}$$

Power Series: Bessel's Equation

- Bessel's equaiton

$$x^2y'' + xy' + (x^2 - \nu^2)y = 0,$$

with a given number ν.

- Bessel's equation appears when a problem shows cylindrical symmetry.

- By using the series

$$y(x) = x^r \sum_{m=0}^{\infty} a_m x^m$$

we have the *indicial equaiton*

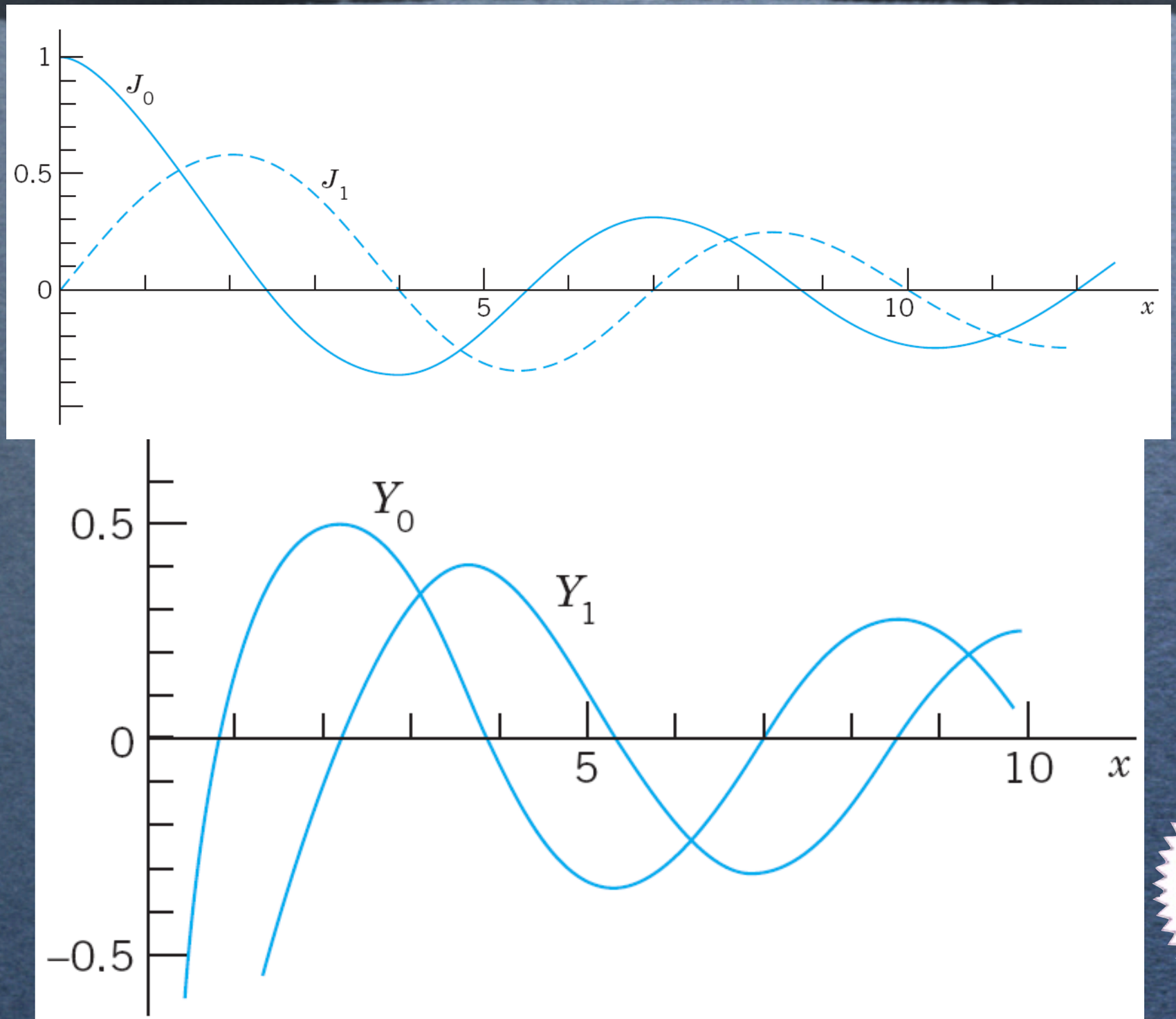
$$(r + \nu)(r - \nu) = 0.$$

- $y(x) = c_1 J_\nu(x) + c_2 Y_\nu(x)$, with two basis solutions

$$J_\nu(x) = x^\nu \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{2^{2m+\nu} m! \Gamma(\nu + m + 1)!}, \qquad \text{Bessel function of the first kind}$$

$$Y_\nu(x) = \frac{1}{\sin \nu\pi}[J_\nu(x) \cos \nu\pi - J_{-\nu}(x)], \qquad \text{Bessel function of the Second kind}$$

Power Series: Bessel's Equation, cont.

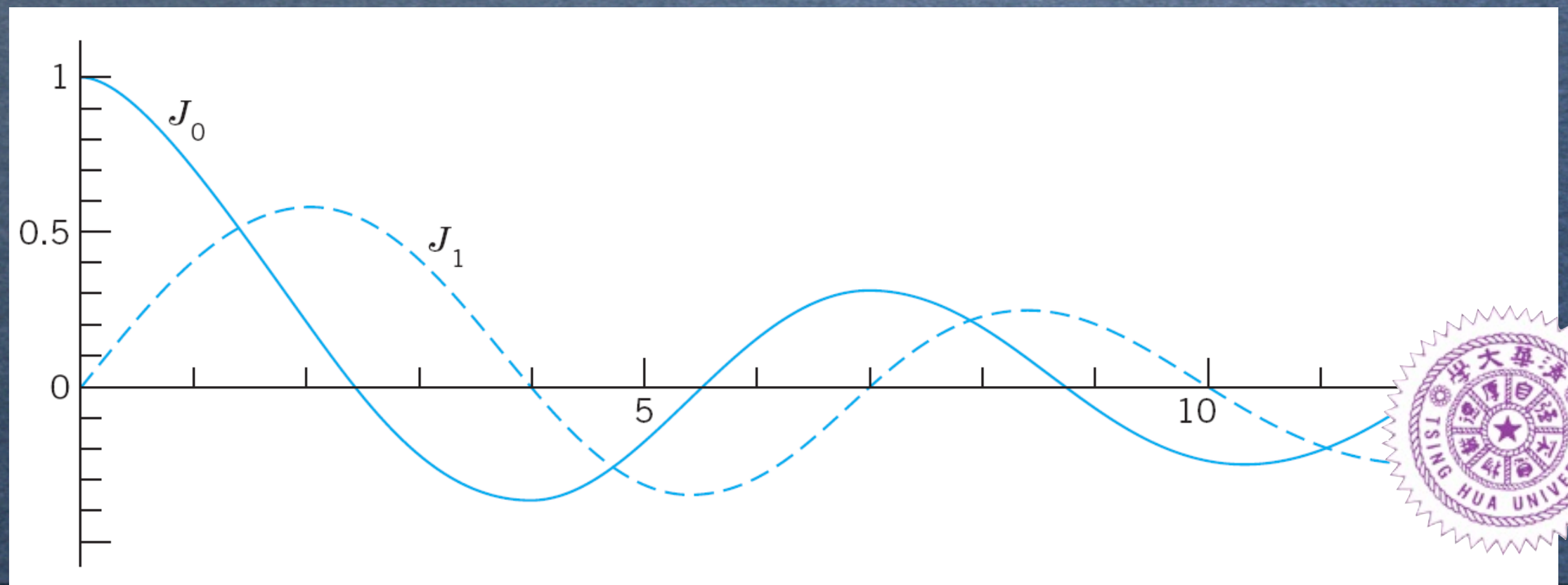


Homework 5: Bessel function

Solve the Bessel's equation, for J_0, $\nu = 0$

$$x^2y'' + xy' + (x^2 - \nu^2)y = 0,$$

with the initial conditions: $J_0(1) = 0.765198$ and $J_0'(1) = -0.440051$ for the range $x : [0, 10]$.



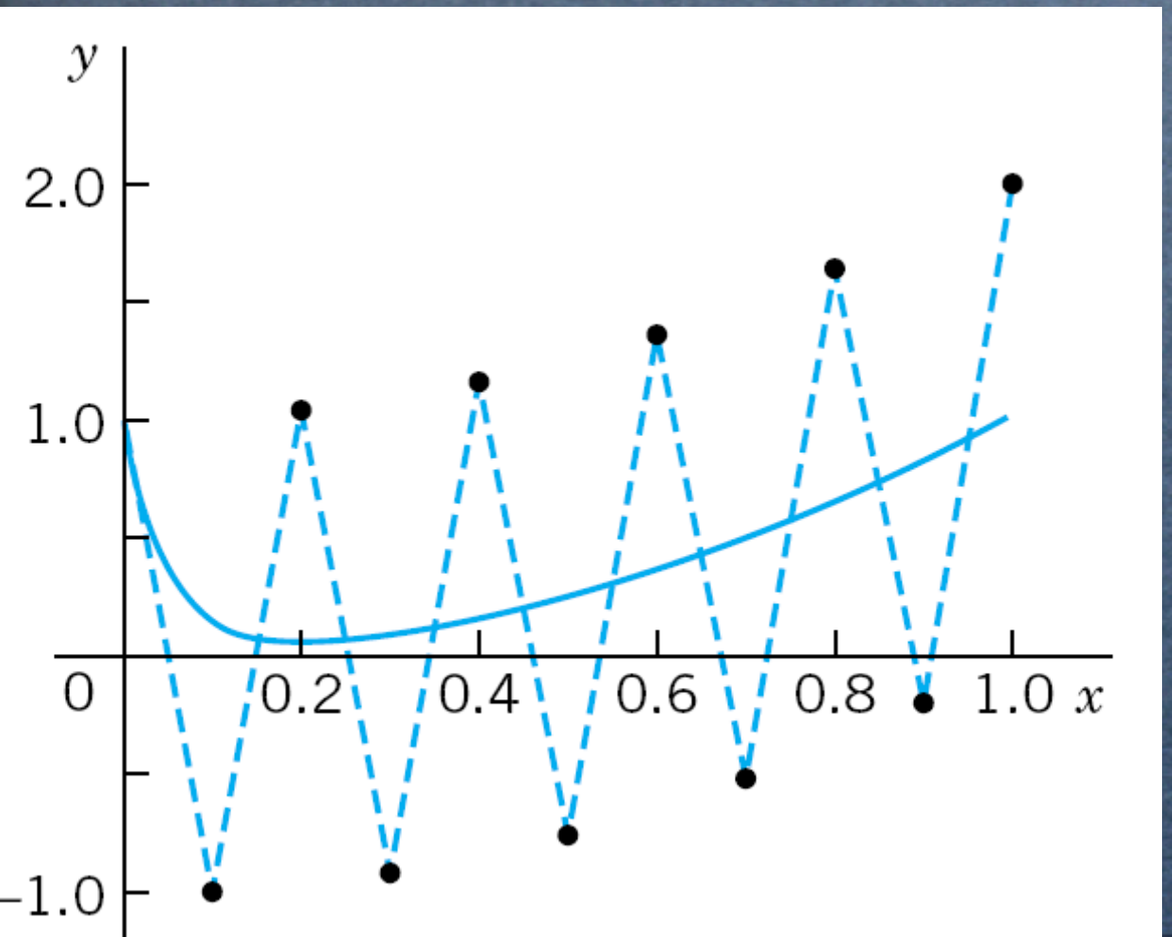
ODE: Stiff problem

The initial value problem:

$$y' = -20\,y + 20\,x^2 + 2\,x, \qquad y(0) = 1,$$

has the solution

$$y = e^{-20x} + x^2.$$



ODE: Backward Euler's method

For a first-order differential equation:

$$y'(t) = f(x, y), \quad \text{with} \quad y(0) = y_0,$$

the backward Euler formula is,

$$y_{n+1} = y_n + h\, f(x_{n+1}, y_{n+1}).$$

ODE: Stiff problem

For the example:

$$\begin{aligned} y_{n+1} &= y_n + h\, f(x_{n+1}, y_{n+1}), \\ &= y_n + h\,(-20\, y_{n+1} + 20\, x_{n+1}^2 + 2\, x_{n+1}), \\ &= \frac{y_n + h[20(x_n + h)^2 + 2(x_n + h)]}{1 + 20\, h}, \qquad x_{n+1} = x_n + h. \end{aligned}$$

ODE: Stiff problem

x	BEM $h = 0.05$	BEM $h = 0.2$	Euler $h = 0.05$	Euler $h = 0.1$	RK $h = 0.1$	RK $h = 0.2$	Exact
0.0	1.00000	1.00000	1.00000	1.00000	1.00000	1.000	1.00000
0.1	0.26188		0.00750	−1.00000	0.34500		0.14534
0.2	0.10484	0.24800	0.03750	1.04000	0.15333	5.093	0.05832
0.3	0.10809		0.08750	−0.92000	0.12944		0.09248
0.4	0.16640	0.20960	0.15750	1.16000	0.17482	25.48	0.16034
0.5	0.25347		0.24750	−0.76000	0.25660		0.25004
0.6	0.36274	0.37792	0.35750	1.36000	0.36387	127.0	0.36001
0.7	0.49256		0.48750	−0.52000	0.49296		0.49001
0.8	0.64252	0.65158	0.63750	1.64000	0.64265	634.0	0.64000
0.9	0.81250		0.80750	−0.20000	0.81255		0.81000
1.0	1.00250	1.01032	0.99750	2.00000	1.00252	3168	1.00000

Homework 5: Stiff problem

$$
\begin{aligned}
x'(t) &= -20\,x - 19\,y, & x(0) &= 2,\\
y'(t) &= -19\,x - 20\,y, & y(0) &= 0,
\end{aligned}
$$

The solutions are

$$
\begin{aligned}
x(t) &= e^{-39\,t} + e^{-t},\\
y(t) &= e^{-39\,t} - e^{-t}.
\end{aligned}
$$

ODE: Implicit integration method

Consider the single equation,

$$y' = -cy$$

where $c > 0$ is a constant. The explicit (or forward) Euler scheme for integrating this equation with stepsize h is

$$y_{n+1} = y_n + hy'_n = (1 - ch)y_n.$$

Clearly this method is unstable if $h > 2/c$, for then $|y_n| \to \infty$ as $n \to \infty$.
The simplest cure is to resort implicit differencing, backward Euler scheme,

$$y_{n+1} = y_n + hy'_{n+1}$$

or

$$y_{n+1} = \frac{y_n}{1 + ch}$$

This method is absolutely stable: even as $h \to \infty$, $y_{n+1} \to 0$, which is the true equilibrium solution.

Implicit integration method, system

For a set of linear equations

$$\mathbf{Y}' = -\mathbf{C} \cdot \mathbf{Y}$$

where $\mathbf{C}$ is a positive definite matrix. Explicit differencing gives

$$\mathbf{Y}_{n+1} = (\mathbf{1} - \mathbf{C}h) \cdot \mathbf{Y}_n.$$

Now a matrix $\mathbf{A}^n$ tends to zero as $n \to \infty$ only if the largest eigenvalue of $\mathbf{A}$ has magnitude less than unity. Thus $\mathbf{Y}_n$ is bounded as $n \to \infty$ only if the largest eigenvalue of $\mathbf{1} - \mathbf{C}h$ is less than 1, or

$$h < \frac{2}{\lambda_{max}}$$

where λ_{max} is the largest eigenvalue of $\mathbf{C}$. Implicit differencing gives

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + h\mathbf{Y}'_{n+1}, \quad \text{or} \quad \mathbf{Y}_{n+1} = (\mathbf{1} + \mathbf{C}h)^{-1} \cdot \mathbf{Y}_n$$

Thus the method is stable for all stepsizes h. The penalty we pay for this stability is that we are required to invert a matrix at each step.

ODE: Semi-implicit integration method

For a nonlinear system

$$\mathbf{Y}' = \mathbf{F}(\mathbf{Y}),$$

where $\mathbf{F}$ is a nonlinear operator, and the implicit differencing gives

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + h\mathbf{F}(\mathbf{Y}_{n+1})$$

In general this is some nasty set of nonlinear equations that hs to be solved iteratively at each step.
Suppose we try linearizing the equations, as in Newton's method:

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + h[\mathbf{F}(\mathbf{Y}_n) + \frac{\partial \mathbf{F}}{\partial \mathbf{Y}}|_{\mathbf{Y}_n} \cdot (\mathbf{Y}_{n+1} - \mathbf{Y}_n)]$$

where $\partial\mathbf{F}/\partial\mathbf{Y}$ is the Jacobian matrix. Then

$$\boxed{\mathbf{Y}_{n+1} = \mathbf{Y}_n + h[\mathbf{1} - h\frac{\partial \mathbf{F}}{\partial \mathbf{Y}}]^{-1} \cdot \mathbf{F}(\mathbf{Y}_n).}$$

Solving implicit methods by Linearization is called a "semi-implicit" method.

ODE: Adaptive stepsize control

There are two approximate solutions from x to $x + 2h$,

1. one step $2h$:

$$y_{2h}(x + 2h) = y_1 + (2h)^5\phi + \mathbf{O}(h^6) + \dots$$

2. 2 steps each of size h:

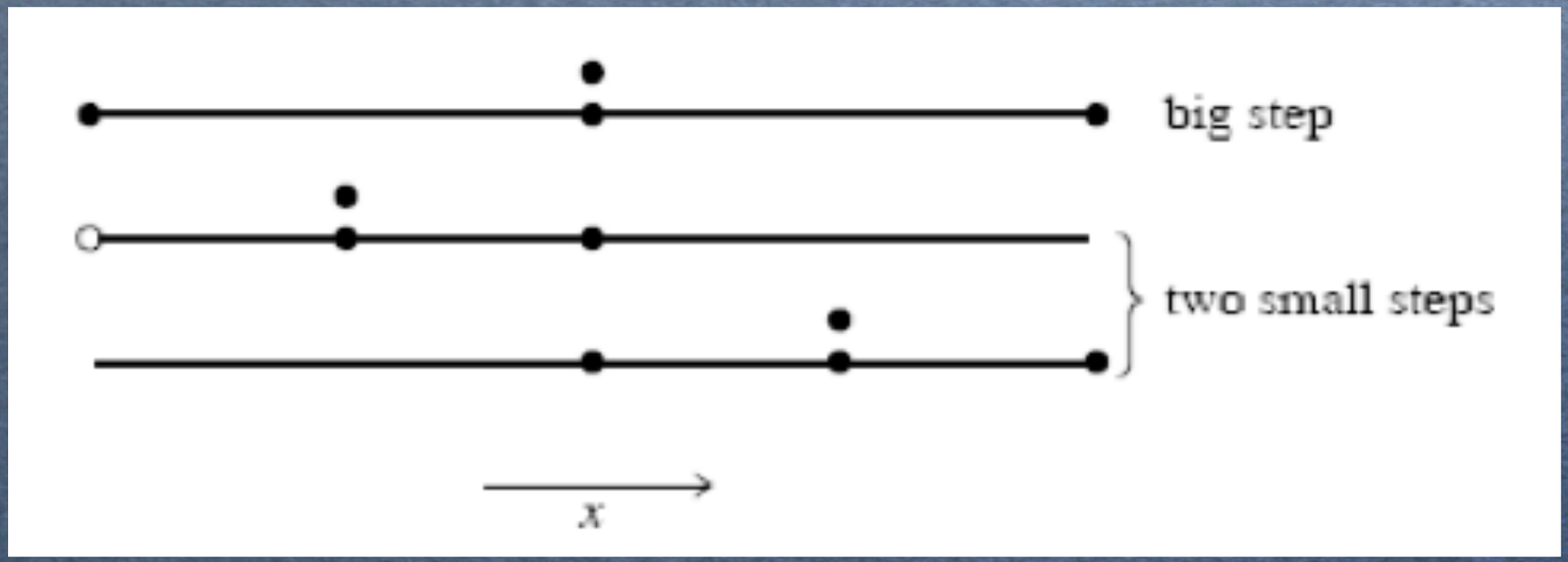
$$y_{2*h}(x + 2h) = y_1 + 2(h)^5\phi + \mathbf{O}(h^6) + \dots$$

The difference between the two numerical estimates is a convenient indicator of truncation error,

$$\boxed{\Delta \equiv y_{2h} - y_{2*h}.}$$

ODE: Adaptive stepsize control

We can keep a desired degree of accuracy by adjusting h with the cost of 11 evolutions (compared to 8).



ODE: Adaptive stepsize control

- Δ scales as h^5.
- If we take a step h_1 and produce an error Δ_1, the next step h_2 is estimated as

$$h_2 = h_1 |\frac{\Delta_2}{\Delta_1}|^2,$$

  where Δ_2 is the desired accuracy.
- If Δ_1 is larger than Δ_2 in magnitude, the equation tells how much to decrease the stepsize.
- If Δ_1 is smaller than Δ_2, the equation tells how much we can safely increase the stepsize for the next step.

ODE: Embedded Runge-Kutta method

The general form of a fifth-order Runge-Kutta formula is

$$k_1 = h\, f(x_n, y_n)$$
$$k_2 = h\, f(x_n + a_2 h, y_n + b_{21} k_1)$$
$$\ldots$$
$$k_6 = h\, f(x_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5)$$

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + \mathbf{O}(h^6)$$

The embedded fourth-order formula is

$$y^*_{n+1} = y_n + c^*_1 k_1 + c^*_2 k_2 + c^*_3 k_3 + c^*_4 k_4 + c^*_5 k_5 + c^*_6 k_6 + \mathbf{O}(h^5)$$

and the error estimate is

$$\Delta \equiv y_{n+1} - y^*_{n+1} = \sum_{i=1}^{6} (c_i - c^*_i) k_i$$

ODE: Predictor-Corrector method

For the ODE, $y' = f(x, y)$, from x_n to x,

$$y(x) = y_n + \int_{x_n}^{x} f(x_1, y)\mathrm{d}x_1$$

- In a single step method, like Runge-Kutta, the value y_{n+1} at x_{n+1} depends only on y_n.
- In a multistep method, we approximate $f(x, y)$ by a polynomial passing through *several* previous points x_n, x_{n-1}, ... and possibly also through x_{n+1}. The result is

$$y_{n+1} = y_n + h[\beta_0 f(x_{n+1}, y_{n+1}) + \beta_1 f(x_n, y_n) + \beta_2 f(x_{n-1}, y_{n-1}) + \cdots]$$

If $\beta_0 = 0$, the method is explicit; otherwise it is implicit.

ODE: Predictor-Corrector method

- Predictor step: for y_{n+1}, two methods suggest, *functional iteration* and *Newton's method*.

1. take some initial guess for y_{n+1},
2. insert it into the right-hand side to get an updated value of y_{n+1} until converges.
3. use some *explicit* formula for the initial guess.

ODE: Predictor-Corrector method

- Corrector step:

  1. In the predictor step we are essentially *extrapolating* the polynomial fit to the derivative from the previous points to the new point x_{n+1}.
  2. Then we doing the integral in a Simpson-like manner from x_n to x_{n+1}.

$$\int_{x_{k-1}}^{x_{k+1}} f(x)\mathrm{d}x \approx \frac{\Delta x}{3}[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})] + \mathbf{O}(\Delta x^5),$$

  3. The subsequent Simpson-like integration, using the prediction step's value of y_{n+1} to *interpolate* the derivative, is called the corrector step.

Adams-Bashforth-Moulton method

Adams-Bashforth-Moulton scheme:

1. Predictor:

$$y_{n+1} = y_n + \frac{h}{12}[23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + f(x_{n-2}, y_{n-2}] + \mathbf{O}(h^4)$$

2. Corrector:

$$y_{n+1} = y_n + \frac{h}{12}[5f(x_{n+1}, y_{n+1}) + 8f(x_n, y_n) - f(x_{n-1}, y_{n-1})] + \mathbf{O}(h^4)$$