Numerical Analysis

Shin, Byeong-Chun

Chonnam National University, Gwangju, Korea

Reference :

1. Elementary Numerical Analysis, Atkinson and Han
2. Numerical Approximation of Partial Differential Equations, . Quarteroni and A. Valli

1 Taylor Polynomials

1.1 The Taylor Polynomial

– Most function $f(x)$ (e.g. $\cos x, e^x, \sqrt{x}$) cannot be evaluated exactly in simple way.

– The most common classes of approximating function $\hat{f}(x)$ are the polynomials and an efficient approximating polynomial is a Taylor polynomial.

– A related form of function is the piecewise polynomial function.

Taylor Polynomial of degree n for a function $f(x)$ about $x = a$:

- linear polynomial, $p_1(x) : p_1(a) = f(a) \quad \text{and} \quad p_1'(a) = f'(a)$

$$p_1(x) = f(a) + (x-a)f'(a).$$

- quadratic polynomial, $p_2(x) : p_2(a) = f(a), \quad p_2'(a) = f'(a) \quad \text{and} \quad p_2''(a) = f''(a)$

$$p_2(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a).$$

- polynomial of degree n, $p_n(x) : p_n(a) = f(a), p_n'(a) = f'(a), \cdots, p_n^{(n)}(a) = f^{(n)}(a)$

$$\left(\text{i.e., } p_n^{(k)}(a) = f^{(k)}(a), \quad k = 0, 1, \cdots, n \right)$$

$$p_n(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + \cdots + \frac{(x-a)^n}{n!} f^{(n)}(a) = \sum_{k=0}^{n} \frac{(x-a)^k}{k!} f^{(k)}(a).$$

Example (1). Find the Taylor polynomial of degree n, $p_n(x; a)$, for $f(x) = e^x$ about $x = a$.

Matlab Code : To compare the three graphs $f(x)$, $p_1(x)$ and $p_2(x)$ around $x = 0$

```
z = -1:0.05:1;                    % z = (x-a) = x
fx = exp(z);                      % function value of f(x)
Dkfzero = ones(1,length(z));      % function value of f^(k)(x) at x=0
p1 = Dkfzero + Dkfzero.*z;        % linear polynomial
p2 = p1 + (1/2)*Dkfzero.*(z.^2); % quadratic polynomial
plot(z,fx,z,p1,z,p2,':')
```

1.2 The Error in Taylor's Polynomial

Theorem 1.1 (Taylor's Remainder). *Assume that $f(x)$ has $n+1$ continuous derivatives on an interval $[\alpha, \beta]$, and let $a \in [\alpha, \beta]$. For the Taylor polynomial $p_n(x)$ of $f(x)$, let $R_n(x) := f(x) - p_n(x)$ denote the remainder in approximating $f(x)$ by $p_n(x)$. Then*

$$R_n(x) = \frac{(x-a)^{n+1}}{(n+1)!} f^{(n+1)}(c_x), \quad \alpha \le x \le \beta$$

with c_x an unknown point between a and x.

Example (2). The approximation error of $f(x) = e^x$ and its Taylor polynomial $p_n(x)$ with $a = 0$ is given by

$$e^x - p_n(x) = R_n(x) = \frac{x^{n+1}}{(n+1)!} e^c \quad (n \ge 0) \quad \text{with } c \text{ between 0 and } x.$$

For each fixed x,

$$e^x - p_n(x) = R_n(x) = \frac{x^{n+1}}{(n+1)!}\, e^c \longrightarrow 0 \quad \text{as} \quad n \longrightarrow \infty \quad \text{because} \quad \lim_{n\to\infty} \frac{|x|^n}{n!} = 0.$$

Let us take the degree n so that $p_n(x)$ approximates $f(x)$ on an interval $[-1, 1]$ with the accuracy

$$|R_n(x)| \le 10^{-9}.$$

Using the upper bound of $|R_n(x)|$

$$|R_n(x)| = \frac{|x|^{n+1}}{(n+1)!}\, e^c \le \frac{e}{(n+1)!} < \frac{3}{(n+1)!} \le 10^{-9},$$

we can find the sufficient degree n so that the approximation error is bounded by the tolerance 10^{-9} :

$$|R_n(x)| \le 10^{-9} \quad \text{when} \quad n \ge 12.$$

Some Taylor polynomials :

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \frac{x^{n+1}}{(n+1)!}\, e^c,$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + (-1)^n \frac{x^{2n+1}}{(2n+1)!} \cos c,$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots + (-1)^n \frac{x^{2n}}{(2n)!} + (-1)^{n+1}\frac{x^{2n+2}}{(2n+2)!} \sin c,$$

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots + x^n + \frac{x^{n+1}}{1-x}, \quad (x \ne 1),$$

$$(1+x)^\alpha = 1 + \binom{\alpha}{1} x + \binom{\alpha}{2} x^2 + \cdots + \binom{\alpha}{n} x^n + \binom{\alpha}{n+1} x^{n+1}(1+c)^{\alpha-n-1},$$

where the binomial coefficients are defined by

$$\binom{\alpha}{k} = \frac{\alpha(\alpha-1)\cdots(\alpha-k+1)}{k!}, \quad k = 1, 2, 3, \cdots .$$

Assume that $f(x)$ is infinitely many differentiable at $x = a$ and

$$\lim_{n\to\infty} [f(x) - p_n(x)] = \lim_{n\to\infty} R_n(x) = 0,$$

the infinite series

$$\sum_{k=0}^{\infty} \frac{(x-a)^k}{k!} f^{(k)}(a)$$

is called the **Taylor series expansion** of the function $f(x)$ about $x = a$.

2 Error and Computer Arithmetic

2.1 Floating-Point Numbers

– Numbers must be stored in computers and arithmetic operations must be performed on these numbers.

– Most computers have two ways of storing numbers, in integer format and in floating-point format.

Floating-Point Representation in the Decimal system : more intuitive

$$x = \sigma \cdot \bar{x} \cdot 10^e$$

where $\sigma = +1$ or -1 (sign), e is an integer (exponent), and $1 \le \bar{x} < 10$ (significant or mantissa).

For an example,

$$124.62 = +1 \cdot (1.2462) \cdot 10^2$$

with the sign $\sigma = +1$, the exponent $e = 2$, and the significant $\bar{x} = 1.2462$.

- The above example is a five-digit decimal floating-point arithmetic.
- The last digit may need to be changed by rounding.

Floating-Point Representation in the Binary system

$$x = \sigma \cdot \bar{x} \cdot 10^e$$

where $\sigma = +1$ or -1 (sign), e is an integer (exponent), and $(1)_2 \le \bar{x} < (10)_2$ is a binary fraction.

For an example,

$$(11011.0111)_2 = (1.10110111)_2 \cdot 2^{(100)_2} \quad \text{with } \sigma = +1,\ e = (100)_2 = 4, \text{ and } \bar{x} = (1.10110111)_2.$$

- The allowable number of binary digits in $\bar{x}$ is called the precision of the binary floating-point representation.

Single Precision Floating-Point Representation of x has a precision of 24 binary digits and uses 4 bytes (32 bits):

$$x = \sigma \cdot (1.b_{10}\ b_{11}\ \cdots\ b_{32}) \cdot 2^e \quad \text{(mantissa of decimal digits is 7 or 8) in normalized format } \bar{x}$$

with the exponent e limited by

$$-126 = -(1111110)_2 \le e \le (1111111)_2 = 127.$$

σ	E	$\bar{x}$
b_1	$b_2\ b_3\ \cdots\ b_9$	$b_{10}\ b_{11}\ \cdots\ b_{32}$

$$-126 \le e(:= E - 127) \le 127 \quad \text{with} \quad 0 \le E = (b_2\, b_3\, \cdots\, b_9)_2 \le 255$$

- But, if $E = (00\cdots0)_2 = 0$, then $e = -126$ and $\bar{x} = (0.b_{10}\ b_{11}\ \cdots\ b_{32})_2$ with unnormalized format $\bar{x}$
- if $E = (11\cdots1)_2 = 255$ and $b_{10} = \cdots = b_{32} = 0$, then $\bar{x} = \pm\infty$,
- if $E = (11\cdots1)_2 = 255$ and $\sim\big(b_{10} = \cdots = b_{32} = 0\ \big)$ then $\bar{x} = NaN$.

Double Precision Floating-Point Representation of x has a precision of 53 binary digits and uses 8 bytes (64 bits):

$$\bar{x} = \sigma \cdot (1.b_{13}\ b_{14}\ \cdots\ b_{64}) \cdot 2^e \qquad \text{with } -1022 \le e \le 1023 \qquad (\text{mantissa of decimal digits is 15 or 16})$$

σ	$E(:= e + 1023)$	$\bar{x}$
b_1	$b_2\ b_3\ \cdots\ b_{12}$	$b_{13}\ b_{14}\ \cdots\ b_{64}$

2.2 Accuracy of Floating-Point Representation

Machine epsilon is the difference between 1 and the next larger number that can be stored in the floating-point format.

In single precision IEEE format, the next larger binary number is

1.00000000000000000000001

with the final binary digit 1 in position 23 to the right of the binary point.

The machine epsilon in single precision format is $2^{-23} \approx 1.19^{-7}$.

In a similar fashion,

The machine epsilon in double precision format is $2^{-52} \approx 2.22^{-16}$.

- In Matlab, it uses the double precision format so that the machine epsilon is eps $\approx 2.22^{-16}$.

Largest integer M that any integer x $(0 \le x \le M)$ can be stored or represented exactly in floating-point form :

- In the single precision format (24 binary digits) :

$$M = (1.00\cdots0)_2 \cdot 2^{24} = 2^{24} = 16777216 \approx 1.67^{7}.$$

- In the double precision format (53 binary digits) :

$$M = (1.00\cdots0)_2 \cdot 2^{53} = 2^{53} \approx 9.0^{15}.$$

2.2.1 Rounding and Chopping

Let the significant in the floating-point representation contain n binary digits.

If the number x has a significant $\bar{x}$ that requires more than n binary bits, then it must be shortened when x is stored in the computer.

- The simplest method is to simply truncate or chop $\bar{x}$ to n binary digits ignoring the remaining digits.
- The second method is to round $\bar{x}$ to n digits based on the size of the part of $\bar{x}$ following digit n.

  Denote the machine floating-point version of a number x by $\mathrm{fl}(x)$.

  Then $\mathrm{fl}(x)$ can be written in the form

$$\mathrm{fl}(x) = x \cdot (1+\epsilon) \quad \text{with a small number } \epsilon.$$

  Chopping : $-2^{-n+1} \le \epsilon \le 0$

Rounding : $-2^{-n} \leq \epsilon \leq 2^{-n}$: much better

- The IEEE standard is using the rounding :

  Single precision : $-2^{-24} \leq \epsilon \leq 2^{-24}$

  Double precision : $-2^{-53} \leq \epsilon \leq 2^{-53}$

2.2.2 Errors

Denote by x_T the true value and x_A an approximate value.

Error : $\text{Error}(x_A) = x_T - x_A$

Relative Error : $\text{Rel}(x_A) = \dfrac{x_T - x_A}{x_T}$

2.2.3 Sources of Error

Modelling Errors

Blunders and Mistakes

Physical Measurement Errors

Machine Representation and Arithmetic Errors

Mathematical Approximation Errors

2.2.4 Loss-of-Significance Errors

Compare the followings :

$$f(x) = x\left(\sqrt{x+1} - \sqrt{x}\right) \qquad f(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}$$

To avoid the loss of significant digits, use another formulation for $f(x)$, avoiding the subtraction of nearly equal quantities.

2.2.5 Noise in Function Evaluation

Using floating-point arithmetic with rounding or chopping, arithmetic operations (e.g., additions and multiplications) cause errors in the evaluation of $f(x)$, generally quite small ones.

2.3 Underflow and Overflow Errors

Underflow : Attempts to create numbers that are too small lead to what are called underflow errors.

For example, consider an evaluation

$$f(x) = x^{10} \quad \text{for } x \text{ near } 0.$$

In the single precision arithmetic, the smallest nonzero positive number expressible in *normalized* floating-point format (using the form of significant $\bar{x} = (1.a_1 \cdots a_{23})_2$) is

$$m = (1.0 \cdots 0)_2 \cdot 2^{-126} = 2^{-126} \approx 1.18 \times 10^{-38}.$$

Thus $f(x)$ will be set to zero if

$$x^{10} < m \iff |x| < \sqrt[10]{m} \approx 1.61 \times 10^{-4} \iff -0.000161 < x < 0.000161$$

- If the use of *unnormalized* floating-point numbers (using the form of significant $\bar{x} = (0.a_1 \cdots a_{23})_2$) allows to represent the smaller number.

$$m = (0.0 \cdots 1)_2 \cdot 2^{-126} = 2^{-149} \approx 1.4 \times 10^{-45}.$$

- Matlab uses the double precision unnormalized floating-point numbers.

  (Using the form of significant $\bar{x} = (0.a_1 \cdots a_{52})_2$)

$$m = (0.0 \cdots 1)_2 \cdot 2^{-1022} = 2^{-1074} \approx 4.94 \times 10^{-324} \qquad \text{but} \qquad 2^{-1075} = 0 = 10^{-324}.$$

Overflow : Attempts to create numbers that are too large lead to what are called overflow errors (more fatal errors).

In the double precision arithmetic, the largest positive number expressible in floating-point format

$$M = (1.1\cdots1)_2 \cdot 2^{1023} = (1.1\cdots1)_2 \cdot 2^{52} \cdot 2^{971}$$

$$= (11\cdots1)_2 \cdot 2^{971} = (2^{53}-1)\cdot 2^{971} \approx 1.80 \times 10^{308}.$$

It is possible to eliminate an overflow error by just reformulating the expression being evaluated.

For example, with very large x and y (e.g., $x = 10^{200}$ and $y = 10^{150}$), compare the followings

$$z = \sqrt{x^2+y^2} \qquad \text{and} \qquad z = \begin{cases} |x|\sqrt{1+(y/x)^2}, & 0 \le |y| \le |x| \\ |y|\sqrt{1+(x/y)^2}, & 0 \le |x| \le |y|. \end{cases}$$

Summation Add S from smallest to largest terms for the sum

$$S = a_1 + a_2 + \cdots + a_n = \sum_{j=1}^{n} a_j$$

A Loop Error Keep away from successive repeated rounding errors in operations.

The following loop

```
for j = 1:n
    x = a + j*h
end
```

is better than

```
x = a;
for j = 1:n
    x = x + h
end
```

in general.

3 RootFinding

Calculating the roots of an equation $f(x) = 0$

3.1 The Bisection Method

Suppose that $f(x)$ is continuous on an interval $a \le x \le b$ and that

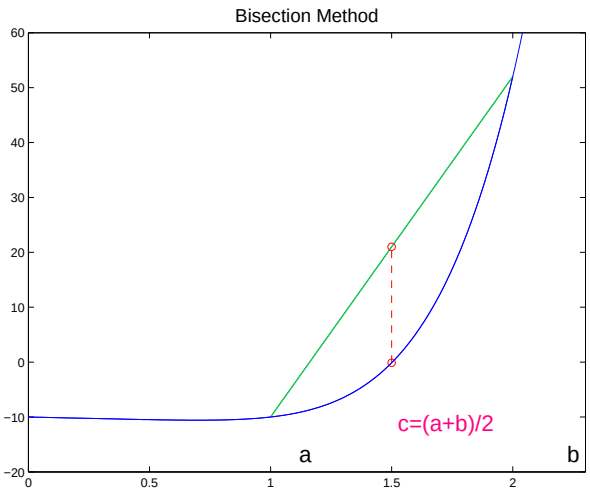
$$f(a)f(b) < 0.$$

Algorithm (Bisection Method) : To find a root to $f(x) = 0$

Input Arguments : function f, Initial guess a and b, Tolerance tol

Output Argument : approximated root c

1. Define $c = (a + b)/2$.
2. If $b - c \le tol$, then accept c as the root and stop.
3. If $sign[f(b)] \cdot sign[f(c)] \le 0$, then set $a = c$.
   Otherwise, set $b = c$.
4. Return to step 1.



Error Bounds

Let a_n, b_n and c_n denote the nth computed values of a, b and c, respectively. Then

$$b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b - a), \quad n \geq 1$$

Since a root α is in either the interval $[a_n, c_n]$ or $[c_n, b_n]$,

$$|\alpha - c_n| \leq c_n - a_n = b_n - c_n = \tfrac{1}{2}(b_n - a_n) = \tfrac{1}{2^n}(b - a). \quad \Leftarrow \text{ linear convergence}$$

Hence,

the iterates c_n converge to α as $n \to \infty$.

How many iterations?

From

$$|\alpha - c_n| \leq \frac{1}{2^n}(b - a) \leq \epsilon,$$

we have

$$n \geq \frac{\log\left(\frac{b-a}{\epsilon}\right)}{\log 2}.$$

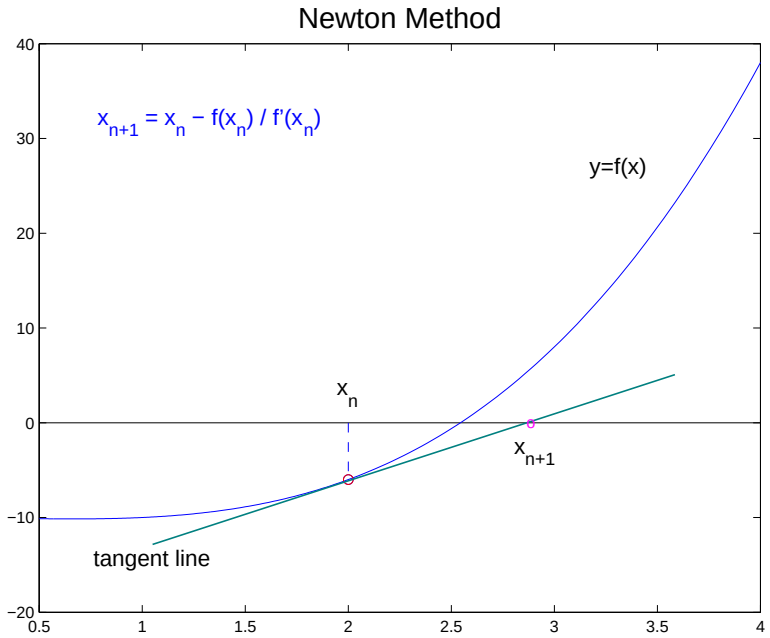
Advantages

- This method guarantees to converge.
- This method generally converges more slowly than most other methods (e.g., for smooth functions).

Matlab Code : **(Bisection Method)**

```
%-------------------------------------------%
function [rt, err] = bisect(a,b,tol,maxitr)
%-------------------------------------------%
  % function [rt, err] = bisect(a,b,tol,maxitr)
  if sign(f(a))*sign(f(b)) > 0
     disp(' f(a) and f(b) are of the same sign.  Stop! '); return
  end
  if a >= b
     tmp = b;  b = a;  a = tmp;   % Make a < b
  end
  c = (a+b)/2;   itr = 0;
  fprintf('\n  itr      a     b     root     f(c)    error    \n')
  while (b-c > tol) & (itr < maxitr)
     itr = itr + 1;
     if sign(f(b))*sign(f(c)) <= 0, a = c;
     else b = c;
     end
     c = (a+b)/2;
     fprintf(' %4.0f  %10.7f  %10.7f  %10.7f  %12.4e  %12.4e \n', ...
                 itr, a, b, c, f(c), b-c);
  end
     rt = c;   err = b - c;
%-------------------------------------------%
function y = f(x)
   y = x.^6 - x - 1;
```

3.2 Newton's Method



Let $y = p_1(x)$ be the linear Taylor polynomial (the tangent line passing through $(x_0, f(x_0))$) of $y = f(x)$ at $x = x_0$:

$$p_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

If x_1 is a root of $p_1(x)$, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Algorithm (Newton's Iteration) : To find a root of $f(x)$

With an initial guess x_0

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, ...$$

Error Bounds

Assume that $f \in C^2$ in some interval about the root α and $f'(\alpha) \neq 0$.

Using Taylor's theorem yields

$$0 = f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{1}{2}(\alpha - x_n)^2 f''(c_n)$$

with c_n an unknown point between α and x_n.

$$0 = \frac{f(x_n)}{f'(x_n)} + \alpha - x_n + (\alpha - x_n)^2 \frac{f''(c_n)}{2f'(x_n)}$$

Using Newton's iteration $\dfrac{f(x_n)}{f'(x_n)} = x_n - x_{n+1}$,

$$0 = x_n - x_{n+1} + \alpha - x_n + (\alpha - x_n)^2 \frac{f''(c_n)}{2f'(x_n)} \quad \text{or} \quad \alpha - x_{n+1} = (\alpha - x_n)^2 \left[-\frac{f''(c_n)}{2f'(x_n)} \right].$$

Thus, we have the second order error estimates (quadratic convergence):

$$M|\alpha - x_n| \leq |M(\alpha - x_{n-1})|^2 \leq \cdots \leq |M(\alpha - x_0)|^{2^n} \quad \text{where} \quad \left| \frac{f''(c_n)}{2f'(x_n)} \right| \leq M.$$

- If the initial error is sufficiently small, then the error in the succeeding iterate will decrease very rapidly.

Example. Find a root $f(x) = x^6 - x - 1$ with an initial guess $x_0 = 1.5$ and then compare this with the results for the bisection method.

Example (**How to compute $\frac{1}{b}$ without the division?**).

Assume $b > 0$. Let $f(x) = b - \frac{1}{x}$. Then the root is $\alpha = \frac{1}{b}$.

Using the derivative $f'(x) = \frac{1}{x^2}$, we have the Newton Method given by

$$x_{n+1} = x_n - \frac{b - \frac{1}{x_n}}{\frac{1}{x_n^2}} \quad \text{or} \quad x_{n+1} = x_n(2 - bx_n).$$

This involves only multiplication and subtraction.

For the error, using $\alpha = \frac{1}{b}$ yields

$$Rel(x_n) = [Rel(x_{n-1})]^2 = \cdots = [Rel(x_0)]^{2^n} \qquad (n \geq 0).$$

The Newton iteration converges to $\alpha = \frac{1}{b}$ with the second order if and only if

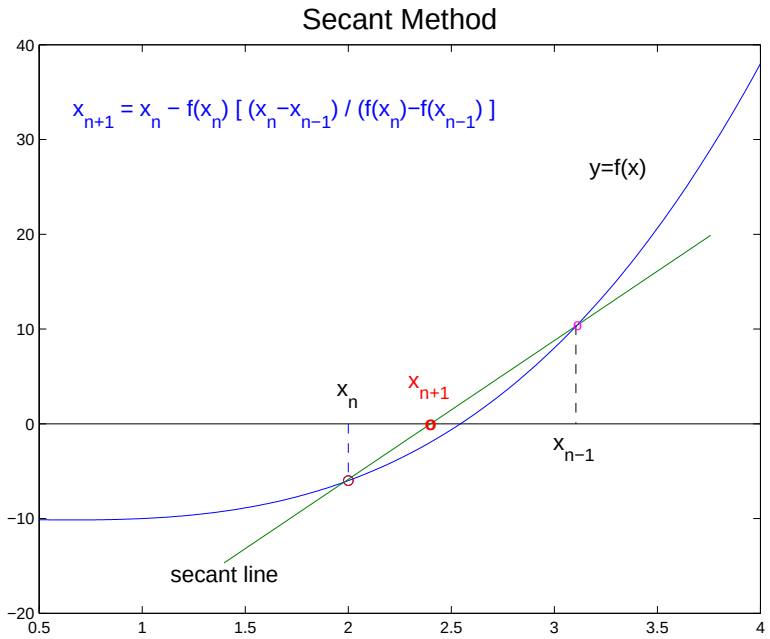
$$|Rel(x_0)| < 1 \iff -1 < \frac{\frac{1}{b} - x_0}{\frac{1}{b}} < 1 \iff 0 < x_0 < \frac{2}{b}.$$

If $|Rel(x_0)| = 0.1$, then $|Rel(x_n)| = 10^{-2^n}$. (e.g., $|Rel(x_4)| = 10^{-16}$)

What is the first order error?

$$|e_n| = \gamma\,|e_{n-1}| = \cdots = \gamma^n\,|e_0| \quad \text{with some } 0 < \gamma < 1.$$

3.3 Secant Method



Let $y = p(x)$ be the linear polynomial (the secant line) passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$:

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

If x_2 is the root of $p(x)$, then $x_2 = x_1 - f(x_1)\dfrac{x_1 - x_0}{f(x_1 - f(x_0)}$.

Algorithm (Secant Method) : To find a root of $f(x)$

With two initial guesses x_0 and x_1

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \qquad n \geq 1.$$

Lagrange Interpolation Polynomial Approximation

If $f \in C^{n+1}$ in some interval $[a,b]$ and $\{x_k\}_{k=0}^n \subset [a,b]$, then there exists a number $\xi(x) \in (a,b)$ such that

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)\cdots(x-x_n)$$

with the Lagrange interpolation and the k-th Lagrange shape polynomial of degree n :

$$P_n(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x) \quad \text{and} \quad L_{n,k}(x) = \prod_{i=0,i\neq k}^{n} \frac{(x-x_i)}{(x_k-x_i)} = \frac{(x-x_0)}{(x_k-x_0)}\cdots\frac{(x-x_n)}{(x_k-x_n)}.$$

Error Bounds

Assume that $f \in C^2$ in some interval about the root α and $f'(\alpha) \neq 0$.

Using Lagrange interpolation yields

$$f(x) = \frac{x-x_{n-1}}{x_n-x_{n-1}}f(x_n) + \frac{x-x_n}{x_{n-1}-x_n}f(x_{n-1}) + \frac{1}{2}(x-x_{n-1})(x-x_n)f''(\xi)$$

and then

$$0 = f(\alpha) = \frac{\alpha-x_{n-1}}{x_n-x_{n-1}}f(x_n) + \frac{\alpha-x_n}{x_{n-1}-x_n}f(x_{n-1}) + \frac{1}{2}(\alpha-x_{n-1})(\alpha-x_n)f''(\xi_n)$$

with ξ_n an unknown point between $\min\{\alpha, x_{n-1}, x_n\}$ and $\max\{\alpha, x_{n-1}, x_n\}$.

Also, we have from the Mean Value theorem that

$$f(x_{n-1}) = f(x_n) - (x_n - x_{n-1})f'(\zeta_n)$$

with ζ_n an unknown point between x_{n-1} and x_n.

From the last two equations, we have

$$0 = f(x_n) + (\alpha - x_n)f'(\zeta_n) + \frac{1}{2}(\alpha - x_{n-1})(\alpha - x_n)f''(\xi_n)$$

or

$$0 = \frac{f(x_n)}{f'(\zeta_n)} + (\alpha - x_n) + (\alpha - x_{n-1})(\alpha - x_n)\frac{f''(\xi_n)}{2f'(\zeta_n)}.$$

Now, using the secant iteration

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} = x_n - \frac{f(x_n)}{f'(\zeta_n)},$$

$$0 = (\alpha - x_{n+1}) + (\alpha - x_{n-1})(\alpha - x_n)\frac{f''(\xi_n)}{2f'(\zeta_n)}$$

so that

$$\alpha - x_{n+1} = (\alpha - x_{n-1})(\alpha - x_n)\left[\frac{-f''(\xi_n)}{2f'(\zeta_n)}\right].$$

Let us consider the following identity

$$|e_{n+1}| \le |e_n| \cdot |e_{n-1}|\, M \quad \text{with} \quad M \approx \left|\frac{-f''(\alpha)}{2f'(\alpha)}\right| \approx \left|\frac{-f''(\xi_n)}{2f'(\zeta_n)}\right|, \quad \forall\, n.$$

Then

$$\frac{|e_{n+1}|}{|e_n|^\gamma} \le M \cdot \left(\frac{|e_n|}{|e_{n-1}|^\gamma}\right)^\alpha \quad \text{with} \quad \gamma = \frac{1+\sqrt{5}}{2}, \quad \alpha = \frac{1-\sqrt{5}}{2}, \ (i.e., \alpha + \gamma = 1,\ \alpha\gamma = -1)$$

so that

$$M^\beta \frac{|e_{n+1}|}{|e_n|^\gamma} \le \left(M^\beta \frac{|e_n|}{|e_{n-1}|^\gamma}\right)^\alpha \le \left(M^\beta \frac{|e_1|}{|e_0|^\gamma}\right)^{\alpha^n} \quad \text{with} \quad \beta = \frac{1}{\alpha - 1}.$$

Taking the limit

$$\lim_{n\to\infty}\frac{|e_{n+1}|}{|e_n|^\gamma} \le M^{-\beta}\lim_{n\to\infty}\left(M^\beta\frac{|e_1|}{|e_0|^\gamma}\right)^{\alpha^n} = M^{-\beta} = M^{1/\gamma} = M^{\gamma-1} \quad \text{because} \quad \lim_{n\to\infty}\alpha^n = 0,$$

we have the following approximate error estimates:

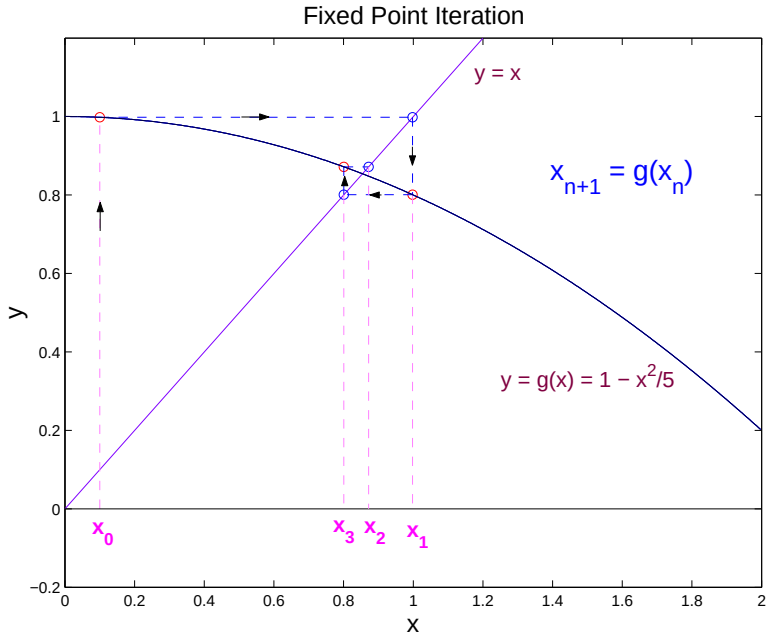
$$|e_{n+1}| = |\alpha - x_{n+1}| \approx c\,|\alpha - x_n|^\gamma = c\,|e_n|^\gamma \quad \text{with} \quad \gamma = \frac{1+\sqrt{5}}{2} \approx 1.62.$$

- If the initial error is sufficiently small, then the error in the succeeding iterate will decrease very rapidly.
- Newton method converges more rapidly than the secant method but it requires two function evaluations per iteration, that of $f(x_n)$ and $f'(x_n)$.
- And the secant method requires only one evaluation $f(x_n)$.
- For many problems with a complicated $f'(x)$, the secant method will probably be faster in actual running time on a computer.

Example. Find a root $f(x) = x^6 - x - 1$ with initial guesses $x_0 = 1$ and $x_1 = 1.5$ using the Secant method, and then compare this with the results for the Newton method.

3.4 Fixed Point Iteration

To find a fixed-point α of $g(x)$: $\alpha = g(\alpha)$



Algorithm (Fixed-Point Iteration) : To find a root α of $x = g(x)$

With an initial guess x_0

$$x_{n+1} = g(x_n), \quad n \geq 0.$$

- If the sequence x_n converges, then $\lim_{n\to\infty} x_n = \alpha$.

Lemma 3.1. *Let $g(x)$ be a continuous function and suppose g satisfies*

$$a \le g(x) \le b \quad \text{for} \quad a \le x \le b.$$

Then, the equation $x = g(x)$ has at least one solution $\alpha \in [a, b]$.

Proof. Define $f(x) = x - g(x)$. Then the function $f(x)$ is continuous on $a \le x \le b$. Since $f(a) \le 0$ and $f(b) \ge 0$, by Intermediate Value Theorem there exists a root of $f(x)$ on $[a, b]$. Hence the equation $x = g(x)$ has at least on solution $\alpha \in [a, b]$. □

Theorem 3.2 (Contraction Mapping Theorem : 축소사상정리)**.** *Assume $g(x)$ and $g'(x)$ are continuous on $[a, b]$, and assume $a \le g(x) \le b$ for $a \le x \le b$. Further assume that*

$$\lambda := \max_{a \le x \le b} |g'(x)| < 1.$$

Then

S1. There is a unique solution α of $x = g(x)$ in the interval $[a, b]$.

S2. For any initial guess $x_0 \in [a, b]$, x_n converges to α.

S3. $|\alpha - x_n| \le \dfrac{\lambda^n}{1 - \lambda} |x_1 - x_0|, \quad (n \ge 0).$

S4. $\lim\limits_{n \to \infty} \dfrac{\alpha - x_{n+1}}{\alpha - x_n} = g'(\alpha) \quad$ *so that* $\quad \alpha - x_{n+1} \approx g'(\alpha)(\alpha - x_n).$

Proof. For $u, v \in [a, b]$, using Mean Value Theorem yields

$$|g(u) - g(v)| = |g'(w)|\,|u - v| \le \lambda\,|u - v|, \quad \text{for some } w \text{ between } u \text{ and } v.$$

S1] Let α and β be solutions to $x = g(x)$. Then $\alpha = g(\alpha)$ and $\beta = g(\beta)$ so that

$$|\alpha - \beta| = |g(\alpha) - g(\beta)| \le \lambda|\alpha - \beta| \quad \text{or} \quad (1-\lambda)|\alpha - \beta| \le 0.$$

Since $\lambda < 1$, $\alpha = \beta$. Hence there exists a unique solution of $x = g(x)$.

S2] By assumption, if $x_0 \in [a, b]$, $x_1 = g(x_0) \in [a, b]$. By mathematical induction, it holds that the fixed-point iterations $x_{n+1} = g(x_n) \in [a, b]$. For convergence of x_n, we have

$$\alpha - x_{n+1} = g(\alpha) - g(x_n) = g'(c_n)(\alpha - x_n) \quad \text{for some } c_n \text{ between } \alpha \text{ and } x_n.$$

Therefore we have

$$|\alpha - x_n| \le \lambda|\alpha - x_{n-1}| \le \lambda^n|\alpha - x_0|, \quad n \le 0.$$

Since $\lambda < 1$, x_n converges to α.

S3] Since

$$|\alpha - x_0| \le |\alpha - x_1| + |x_1 - x_0| \le \lambda|\alpha - x_0| + |x_1 - x_0|,$$

$$(1-\lambda)|\alpha - x_0| \le |x_1 - x_0| \quad \text{or} \quad |\alpha - x_0| \le \frac{1}{1-\lambda}|x_1 - x_0|.$$

Using the argument of S2], we have

$$|\alpha - x_n| \le \frac{\lambda^n}{1-\lambda}|x_1 - x_0|, \quad (n \ge 0).$$

S4] Using the argument of S2 that

$$\alpha - x_{n+1} = g(\alpha) - g(x_n) = g'(c_n)(\alpha - x_n) \quad \text{for some } c_n \text{ between } \alpha \text{ and } x_n$$

we have

$$\lim_{n\to\infty} \frac{\alpha - x_{n+1}}{\alpha - x_n} = \lim_{n\to\infty} g'(c_n).$$

Since c_n is between α and x_n, and since $x_n \to \alpha$, we have $c_n \to \alpha$.

From the continuous of $g'(x)$, we have $g'(c_n) \to g'(\alpha)$. It completes the conclusion.

□

Corollary 3.3. *Assume $\alpha \in (c, d)$. If $g(x)$ and $g'(x)$ are continuous in (c, d), and if*

$$|g'(\alpha)| < 1,$$

then, there is an interval $[a, b]$ around α for which the conclusions of the above theorem are true.

If $g'(\alpha) > 1$, then the fixed-point iterations do not converge.

Error Bounds

We have the linear convergence error bound :

$$|\alpha - x_{n+1}| = \lambda\,|\alpha - x_n| \quad \text{so that} \quad |\alpha - x_n| = \lambda^n\,|\alpha - x_0|.$$

Remark (Order of convergence p)**.**

$$|\alpha - x_{n+1}| = c\,|\alpha - x_n|^p, \quad n \geq 0.$$

Aitken Error Estimation : 오차추정치 : to estimate the error

Denote by $\lambda = g'(\alpha)$. Then

$$\alpha - x_n \approx \lambda(\alpha - x_{n-1})$$

or

$$\alpha \approx x_n + \frac{\lambda}{1-\lambda}(x_n - x_{n-1})$$

Denote by

$$\lambda_n := \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}} = \frac{g(x_{n-1}) - g(x_{n-2})}{x_{n-1} - x_{n-2}} = g'(c_{n-1}) \to g'(\alpha) = \lambda\ .$$

Then we have the Aitken's extrapolation formula : 에이킨의 외간공식 :

$$\alpha - x_n \approx \frac{\lambda_n}{1-\lambda_n}(x_n - x_{n-1}).$$

Example. Find a root $f(x) = 1 - \frac{1}{5}x^2$ using the fixed point method. That is, find the fixed point to $x = g(x) := 1 + x - \frac{1}{5}x^2$.

4 Interpolation and Approximation

4.1 Polynomial Interpolation

4.1.1 Lagrange Interpolation Polynomials

Definition (Interpolation)**.** Let V be a function space with domain D and let $f(x)$ be a function defined on $\{x_i\}_{i=0}^m \subset D$. If a function $p(x) \in V$ satisfies

$$p(x_i) = f(x_i) \text{ for all } i = 0, 1, \cdots, m,$$

then $p(x)$ is called an **interpolation in V for the function $f(x)$ with respect to $\{x_i\}_{i=0}^m$**.

- If V is a space of polynomials, then $p(x)$ is called a polynomial interpolation.

Define the k-th Lagrange shape polynomial (Lagrange polynomial basis) of degree n :

$$L_{n,k}(x) = \prod_{i=0, i\neq k}^{n} \frac{(x - x_i)}{(x_k - x_i)} = \frac{(x - x_0)}{(x_k - x_0)} \cdots \frac{(x - x_{k-1})}{(x_k - x_{k-1})} \frac{(x - x_{k+1})}{(x_k - x_{k+1})} \cdots \frac{(x - x_n)}{(x_k - x_n)}.$$

When we know the degree n apparently, we denote by $L_k(x) := L_{n,k}(x)(x)$.

1. Linear Interpolation which interpolates the values y_i at the points x_i, $i = 0, 1$:

$$P_1(x) = L_{1,0}(x)\, y_0 + L_{1,1}(x)\, y_1 = \frac{x - x_1}{x_0 - x_1}\, y_0 + \frac{x - x_0}{x_1 - x_0}\, y_1.$$

2. Quadratic Interpolation which interpolates the values y_i at the points x_i, $i = 0, 1, 2$:

$$\begin{aligned} P_2(x) &= L_{2,0}(x)\, y_0 + L_{2,1}(x)\, y_1 + L_{2,2}(x)\, y_2 \\ &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\, y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\, y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\, y_2. \end{aligned}$$

3. Higher Degree Interpolation which interpolates the values y_i at the points x_i, $i = 0, 1, \cdots, n$:

$$P_n(x) = \sum_{k=0}^{n} L_{n,k}(x)\, y_k = L_{n,0}(x)\, y_0 + L_{n,1}(x)\, y_1 + \cdots + L_{n,n}(x)\, y_n.$$

Lagrange Polynomial Interpolation of degree n which interpolates $f(x)$ with respect to $\{x_i\}_{i=0}^n$

$$P_n(x) = \sum_{k=0}^{n} L_k(x)\, f(x_k) = L_0(x)\, f(x_0) + L_1(x)\, f(x_1) + \cdots + L_n(x)\, f(x_n).$$

$$P_n(x_i) = f(x_i) \text{ for all } i = 0, 1, \cdots, n.$$

Error Estimate

If $f \in C^{n+1}$ in some interval $[a, b]$ and $\{x_k\}_{k=0}^n \subset [a, b]$, then there exists a number c_x between $\min\{x_i, x\}$ and $\max\{x_i, x\}$ such that

$$f(x) = P_n(x) + \frac{f^{(n+1)}(c_x)}{(n+1)!}(x-x_0)\cdots(x-x_n).$$

Matlab Code : **(Polynomial Interpolation)**

```
------------------------------------------------------------------
>> help polyfun
%%    => interp*, griddata, polyfit, etc

%% Example 1
   x = 0:10; y = sin(x); xi = 0:.25:10;
   yi = interp1(x,y,xi); plot(x,y,'o',xi,yi)

%% Example 2
   x = 0:10; y = sin(x); xi = 0:.25:10;
   p = polyfit(x,y,5);
   pv = polyval(p,xi);
   plot(xi,sin(xi),xi,pv,'-o')

%% Example 3
   rand('seed',0)
   x = rand(100,1)*4-2; y = rand(100,1)*4-2; z = x.*exp(-x.^2-y.^2);
   ti = -2:.25:2;
   [xi,yi] = meshgrid(ti,ti);
   zi = griddata(x,y,z,xi,yi);
   mesh(xi,yi,zi), hold on, plot3(x,y,z,'o'), hold off
------------------------------------------------------------------
```

4.1.2 Divided Difference

Goal : More easily calculable formulation for Interpolation polynomials

The first-order divided difference of $f(x)$

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Then, by the mean value theorem

$$f[x_0, x_1] = f'(c) \quad \text{for some } c \text{ between } x_0 \text{ and } x_1.$$

Also, if x_0 and x_1 are close together, then

$$f[x_0, x_1] \approx f'\left(\frac{x_0 + x_1}{2}\right).$$

The second-order divided difference of $f(x)$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}.$$

The third-order divided difference of $f(x)$

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0}.$$

The nth-order divided difference (Newton divided difference) of $f(x)$

$$f[x_0, x_1, \cdots, x_n] = \frac{f[x_1, \cdots, x_n] - f[x_0, \cdots, x_{n-1}]}{x_n - x_0}.$$

- The values of $f[x_{i0}, x_{i1}, \cdots, x_{in}]$ are same for any permutation $(i_0, i_1, \cdots, i_n)$.

Theorem 4.1. *Let $n \geq 1$, and $f(x) \in C^n([\alpha, \beta])$. Let $x_0, x_1, \cdots, x_n$ be the $n+1$ distinct numbers in $[\alpha, \beta]$. Then*

$$f[x_0, x_1, \cdots, x_n] = \frac{1}{n!} f^{(n)}(c) \quad \textit{for some } \min\{x_i\} < c < \max\{x_i\}.$$

For example, $f[x_0, x_1] = f'(c)$, $f[x_0, x_1, x_2] = \frac{1}{2} f''(c)$, $f[x_0, x_1, x_2, x_3] = \frac{1}{6} f'''(c)$.

Example. Let $f(x) = \cos x$, $x_0 = 0.2$, $x_1 = 0.3$, $x_2 = 0.4$. Check the above Theorem.

Let $P_n(x)$ denote the polynomial interpolation of $f(x)$ at $\{x_i\}_{i=0}^n$:

$$P_n(x_i) = f(x_i) \quad \text{for all } i = 0, 1, \cdots, n.$$

Then, the interpolation polynomial $P_n(x)$ can be written as

Newton divided difference of $f(x)$ of degree n

$$\begin{aligned} P_n(x) &= f(x_0) + (x - x_0) f[x_0, x_1] + \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1}) f[x_0, x_1, \cdots, x_n] \\ &= P_{n-1}(x) + (x - x_0)(x - x_1) \cdots (x - x_{n-1}) f[x_0, x_1, \cdots, x_n]. \end{aligned}$$

For example,

$$\begin{aligned} P_1(x) &= f(x_0) + (x - x_0) f[x_0, x_1], \\ P_2(x) &= f(x_0) + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] \\ &= P_1(x) + (x - x_0)(x - x_1) f[x_0, x_1, x_2]. \end{aligned}$$

TABLE 1. Table of Divided Difference

x_i	$f(x_i)$	First DD	Second DD	Third DD
x_0	$f[x_0]$			
	$\gg$	$f[x_0,x_1]=\dfrac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_1	$f(x_1)$	$\gg$	$f[x_0,x_1,x_2]=\dfrac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
	$\gg$	$f[x_1,x_2]=\dfrac{f[x_2]-f[x_1]}{x_2-x_1}$	$\gg$	$f[x_0,x_1,x_2,x_3]=\dfrac{f[x_1,x_2,x_3]-f[x_0,x_1,x_2]}{x_3-x_0}$
x_2	$f(x_2)$	$\gg$	$f[x_1,x_2,x_3]=\dfrac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	
	$\gg$	$f[x_2,x_3]=\dfrac{f[x_3]-f[x_2]}{x_3-x_2}$	$\gg$	$f[x_1,x_2,x_3,x_4]=\dfrac{f[x_2,x_3,x_4]-f[x_1,x_2,x_3]}{x_4-x_1}$
x_3	$f(x_3)$	$\gg$	$f[x_2,x_3,x_4]=\dfrac{f[x_3,x_4]-f[x_2,x_3]}{x_4-x_2}$	
	$\gg$	$f[x_3,x_4]=\dfrac{f[x_4]-f[x_3]}{x_4-x_3}$	$\gg$	$f[x_2,x_3,x_4,x_5]=\dfrac{f[x_3,x_4,x_5]-f[x_2,x_3,x_4]}{x_5-x_2}$
x_4	$f(x_4)$	$\gg$	$f[x_3,x_4,x_5]=\dfrac{f[x_4,x_5]-f[x_3,x_4]}{x_5-x_3}$	
	$\gg$	$f[x_4,x_5]=\dfrac{f[x_5]-f[x_4]}{x_5-x_4}$		
x_5	$f[x_5]$			

Matlab Code : **(Evaluating Divided Differences)**

```
% Filename  divdif.m
% function dd = divdif(x,f)
%   x(i) = nodes,   f(i) = f(x_i)
%   dd(i) = f[x_0,x_1,\cdots,x_i] : divided differences

function dd = divdif(x,f)
   dd = f;
   m = length(x);
   for i=2:m
      for j=m:-1:i
         dd(j) = ( dd(j)-dd(j-1) )/( x(j)-x(j-i+1) );
      end
   end

      fprintf('--- Divided Differences of f --- \n');
      fprintf('  i       x_i          f(x_i)           D_i  \n');
      fprintf('-----------------------------------------\n');
   for i=1:m
      fprintf(' %3d  %8.2f  %10.4f   %10.4e \n',i-1,x(i),f(i),dd(i))
   end
      fprintf('-----------------------------------------\n');
```

$$f[x_0,x_1,x_2] \;=>\; (x-x_1)f[x_0,x_1,x_2]+f[x_0,x_1] \;=>\; P_2(x)=(x-x_0)\Big((x-x_1)f[x_0,x_1,x_2]+f[x_0,x_1]\Big)+f[x_0]$$

$$y=dd(3) \;=>\; y=(x-x_1)dd(3)+dd(2) \;=>\; y=(x-x_0)\Big((x-x_1)dd(3)+dd(2)\Big)+dd(1)$$

Matlab Code : **(Interpolation using Divided Differences)**

```
% Filename : interp_dd.m
% function yvalues = interp_dd(nodes, fnodes, xvalues)
%   nodes = interpolation nodes,   fnodes = f(nodes)
%   xvalues = coordinate to find the values of interpolation
%   yvalues = interpolated values

function yvalues = interp_dd(nodes,fnodes,xvalues);
   m = length(nodes);
   dd = divdif(nodes,fnodes);
   yvalues = dd(m)*ones(size(xvalues));

   for i=m-1:-1:1
      yvalues = dd(i) + ( xvalues-nodes(i) ).*yvalues;
   end

   plot(nodes,fnodes,'-o',xvalues,yvalues,'--.');
   title('Interpolation using divided differences')
```

4.2 Error in Polynomial Interpolation

Theorem 4.2. *If $f \in C^{n+1}$ in some interval $[a,b]$ and $\{x_k\}_{k=0}^n \subset [a,b]$, then there exists a number c_x between* $\min\{x_i, x\}$ *and* $\max\{x_i, x\}$ *such that*

$$f(x) = P_n(x) + \frac{f^{(n+1)}(c_x)}{(n+1)!}(x-x_0)\cdots(x-x_n).$$

Example (Linear). Take $f(x) = e^x$ on $[0,1]$ and consider x_0, x_1 on $[0,1]$ satisfying $|x_1 - x_0| = h$. Find the maximum error bound of linear interpolation at x $(x_0 < x < x_1)$ to $f(x)$.

$$f(x) - P_1(x) = \frac{(x-x_0)(x-x_1)}{2}\, e^{c_x} \text{ for some } c_x \text{ between between } \min\{x_i, x\} \text{ and } \max\{x_i, x\}.$$

$$|f(x) - P_1(x)| \le \max_{x_0 \le x \le x_1} \left| \frac{(x-x_0)(x-x_1)}{2}\, e^{x_1} \right| \le \frac{e^{x_1} h^2}{8} \le \frac{e\, h^2}{8}. \quad \text{Why? : Homework}$$

Example (Quadratic). Take $f(x) = e^x$ on $[0,1]$ and consider x_0, x_1, x_2 on $[0,1]$ satisfying $x_1 - x_0 = x_2 - x_1 = h$. Find the maximum error bound of the quadratic interpolation at x $(x_0 < x < x_2)$ to $f(x)$.

$$f(x) - P_2(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{6}\, e^{c_x} \text{ for some } c_x \text{ between between } \min\{x_i, x\} \text{ and } \max\{x_i, x\}.$$

$$|f(x) - P_2(x)| \le \max_{x_0 \le x \le x_2} \left| \frac{(x-x_0)(x-x_1)(x-x_2)}{6}\, e^{x_2} \right| \le \frac{e^{x_2} h^3}{9\sqrt{3}} \le \frac{e\, h^3}{9\sqrt{3}}. \quad \text{Why? : Homework}$$

Another Error Formula Let $P_{n+1}(x)$ be the interpolation of $f(x)$ at $x_0, x_1, \cdots, x_n, t$:

$$p_{n+1}(x) = P_n(x) + (x - x_0) \cdots (x - x_n) f[x_0, x_1, \cdots, x_n, t].$$

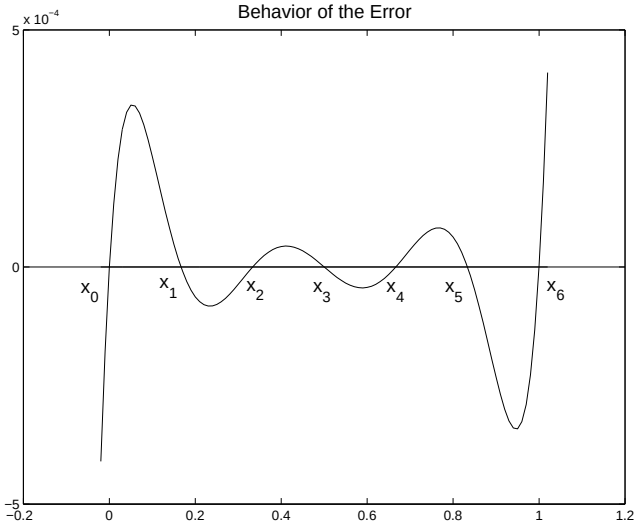
$$f(t) - P_n(t) = p_{n+1}(t) - P_n(t) = (t - x_0) \cdots (t - x_n) f[x_0, x_1, \cdots, x_n, t].$$

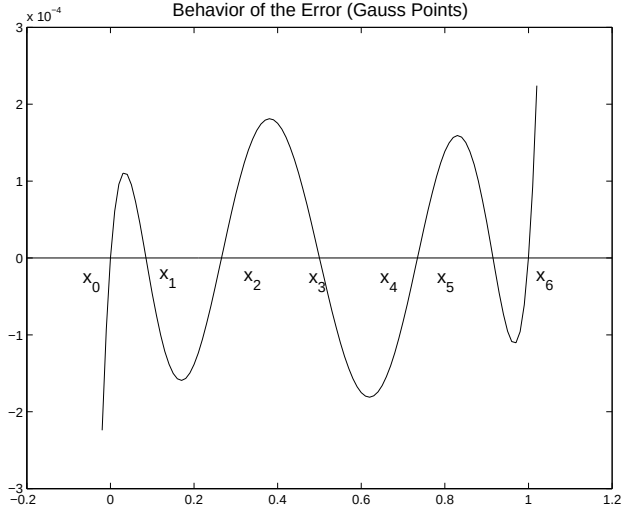
Let

$$\Psi_n(t) = (t - x_0) \cdots (t - x_n).$$

When the node points $x_0, \cdots, x_n$ are evenly space, $\Psi_n(t)$ change greatly through the interval $x_0 \le t \le x_n$.

- The values in $[x_0, x_1]$ and $[x_{n-1}, x_n]$ become much lager than the values in the middle of $[x_0, x_n]$.
- As n increases, this disparity also increases.

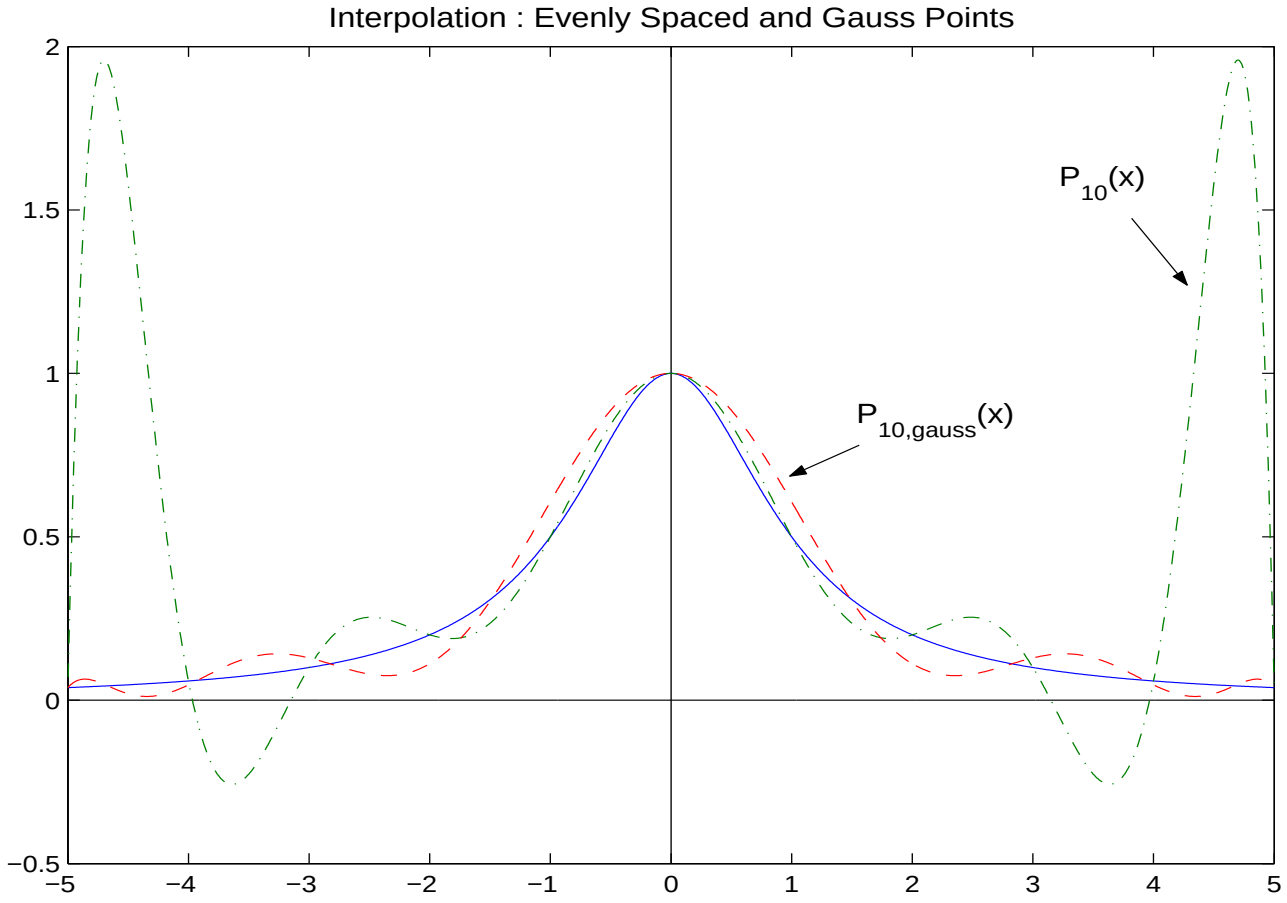




Example (Interpolation). Take $f(x) = \frac{1}{1+x^2}$ on $[-5, 5]$ and let $n > 0$ be an even integer, and define $h = 10/n$

$$x_j = -5 + j * h, \quad j = 0, 1, \cdots, n.$$

consider x_0, x_1, x_2 on $[0, 1]$ satisfying $x_1 - x_0 = x_2 - x_1 = h$. Find the maximum error bound of the quadratic interpolation at x $(x_0 < x < x_2)$ to $f(x)$.

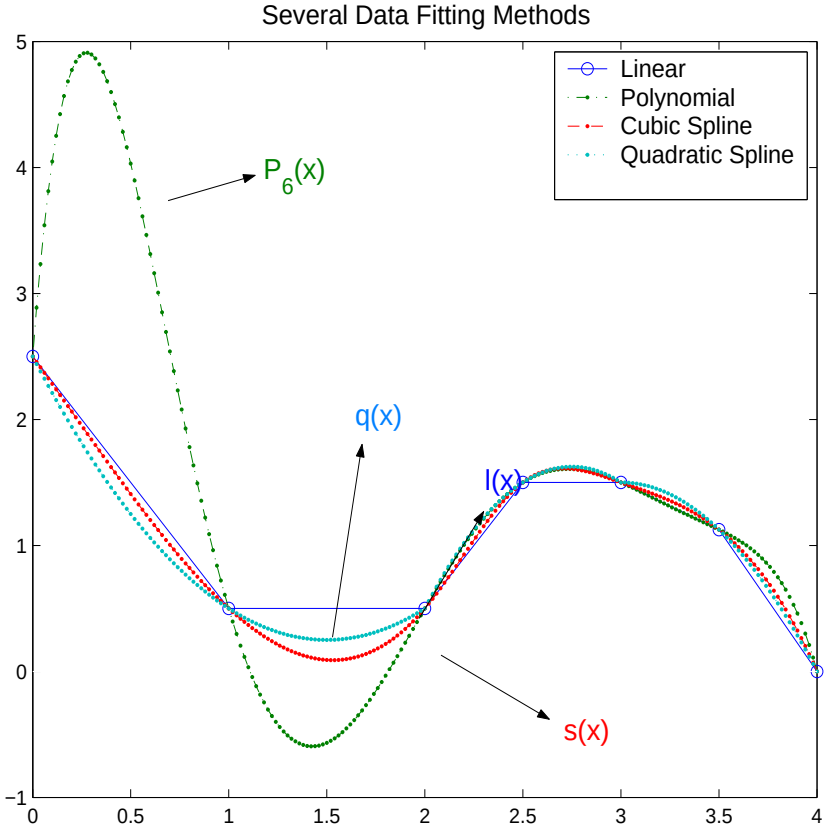


4.3 Interpolation using Cubic Spline Functions

Spline functions are piecewise polynomial functions.

Table. Interpolation Data

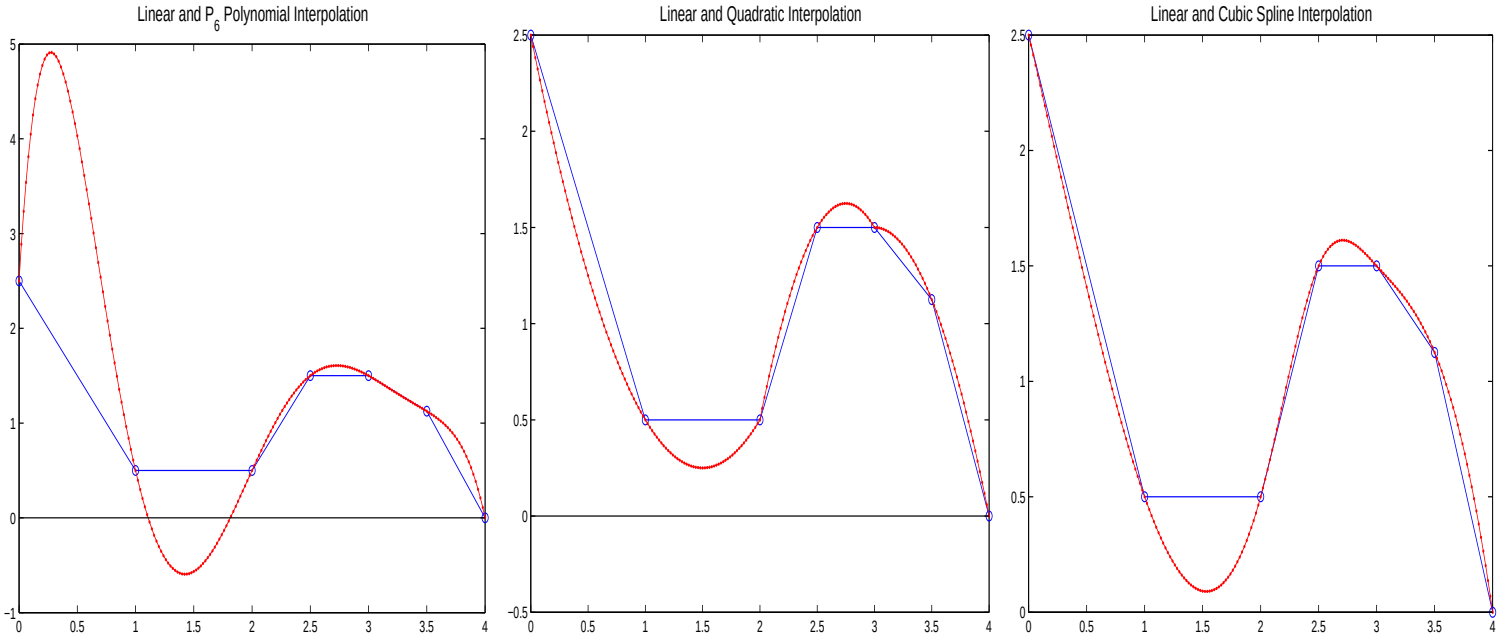
x	0	1	2	2.5	3	3.5	4
y	2.5	0.5	0.5	1.5	1.5	1.125	0



Let $y = l(x)$ be the piecewise linear interpolation.

Goal : We want to construct a smooth curve that interpolates the given data points, but one that follows the shape of $y = l(x)$.

- $y = q(x)$: piecewise quadratic interpolation : not smooth
- $y = p_6(x)$: polynomial interpolation of degree 6 : not similar to $y = l(x)$
- $y = s(x)$: natural cubic spline interpolation function : smooth and similar to $y = l(x)$



4.3.1 Spline Interpolation

Given n data points (x_i, y_i), $i = 1, 2, \cdots, n$ satisfying

$$a = x_1 < x_2 < \cdots < x_n = b,$$

let us seek a function $s(x)$ defined on $[a, b]$ that interpolates the data:

$$s(x_i) = y_i, \quad i = 1, 2, \cdots, n.$$

For smoothness of $s(x)$, we require that $s'(x)$ and $s''(x)$ are continuous.

There is a unique solution $s(x)$ satisfying

- $s(x)$ is polynomial of degree ≤ 3 on each subinterval $[x_{j-1}, x_j]$, $j = 2, 3, \cdots, n$;
- $s(x)$, $s'(x)$, and $s''(x)$ are continuous for $a \leq x \leq b$;
- $s''(x_1) = s''(x_n) = 0$.

This function $s(x)$ is called the natural cubic spline function that interpolates the data $\{(x_i, y_i)\}$.

4.3.2 Construction of the Natural Cubic Spline

Introduce the variables M_i with

$$M_i \equiv s''(x_i), \quad i = 1, 2, \cdots, n.$$

Since $s(x)$ is cubic on each interval $[x_{j-1}, x_j]$ and $s''(x_{j-1}) = M_{j-1}$ and $s''(x_j) = M_j$, we have

$$s''(x) = \frac{(x_j - x)M_{j-1} + (x - x_{j-1})M_j}{x_j - x_{j-1}}, \quad x_{j-1} \leq x \leq x_j.$$

Integrating twice and using the interpolating conditions

$$s(x_{j-1}) = y_{j-1}, \quad s(x_j) = y_j,$$

we have the following cubic polynomial, for $x_{j-1} \le x \le x_j$,

$$s(x) = \frac{(x_j - x)^3 M_{j-1} + (x - x_{j-1})^3 M_j}{6(x_j - x_{j-1})} + \frac{(x_j - x)y_{j-1} + (x - x_{j-1})y_j}{x_j - x_{j-1}} - \frac{1}{6}(x_j - x_{j-1})\big[(x_j - x)M_{j-1} + (x - x_{j-1})M_j\big].$$

To ensure the continuity of $s'(x)$ over $[a, b]$, $s'(x)$ has the same value at their common point $x = x_j$ on each subinterval $[x_{j-1}, x_j]$ and $[x_j, x_{j+1}]$.

This leads to the following system of linear equations:

$$\frac{x_j - x_{j-1}}{6} M_{j-1} + \frac{x_{j+1} - x_{j-1}}{3} M_j + \frac{x_{j+1} - x_j}{6} M_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}, \quad j = 2, 3, \cdots, n-1$$

with $M_1 = M_n = 0$.

Now, we have the so-called tridiagonal system which can be solved by special routines for tridiagonal systems:

$$A\,M = Y$$

where

$$A = \text{diag}\left[\frac{x_j - x_{j-1}}{6}, \ \frac{x_{j+1} - x_{j-1}}{3}, \ + \frac{x_{j+1} - x_j}{6}\right], \quad Y = \left[\frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}\right].$$

Error Estimate

Let $f(x)$ be given on $[a, b]$ and let $n > 1$,

$$h = \frac{b-a}{n-1}, \quad x_j = a + (j-1)h, \quad j = 1, 2, \cdots, n.$$

Let $s_n(x)$ be the natural cubic spline interpolating $f(x)$ at $\{x_j\}$. Then we have

$$\max_{a \le x \le b} |f(x) - s_n(x)| \le C\, h^2$$

where C depends on $f''(a)$, $f''(b)$ and $\max_{a \le x \le b} |f^{(4)}(x)|$.

If $f(x)$ satisfies $f''(a) = f''(b) = 0$, or if we impose the boundary conditions

$$s_n'(a) = f'(a), \quad s_n'(b) = f'(b),$$

or

$$s_n''(a) = f''(a), \quad s_n''(b) = f''(b),$$

then we have the improved error bounds

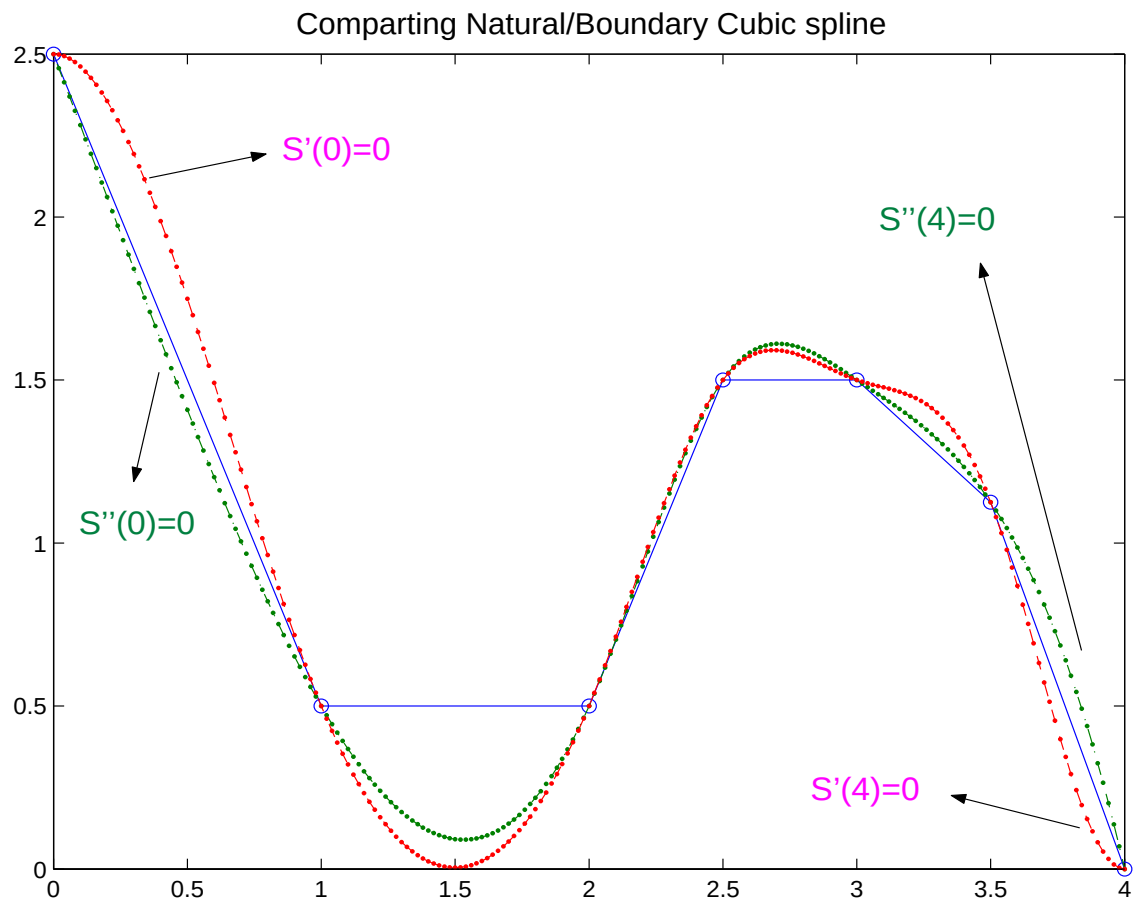
$$\max_{a \le x \le b} |f(x) - s_n(x)| \le C\, h^4.$$

Natural Cubic Spline

sn = spline(x,y,xx)

Cubic Spline with boundary condition $s'(a) = \alpha$, $s'(b) = \beta$

sb = spline(x, [α, y, β], xx)


Comparting Natural/Boundary Cubic spline
S'(0)=0
S''(4)=0
S''(0)=0
S'(4)=0
2.5
2
1.5
1
0.5
0
0
0.5
1
1.5
2
2.5
3
3.5
4

Matlab Code : **(Cubic Spline and Interpolations)**

```
x=[0,1,2,2.5,3,3.5,4];  y=[2.5,0.5,0.5,1.5,1.5,1.125,0];
xx = 0:0.02:4;
%% Linear interpolation
l = interp1(x,y,xx);
%% Polynomial interpolation of degree 6
p6 = polyfit(x,y,6);  p = polyval(p6,xx);
%% Natural Cubic spline interpolation
sn = spline(x,y,xx);
%% Quadratic spline interpolation
x1 = 0:0.02:2; x2 = 2:0.02:3; x3 = 3:0.02:4;
y1 = polyfit(x(1:3),y(1:3),2);  y1 = polyval(y1,x1);
y2 = polyfit(x(3:5),y(3:5),2);  y2 = polyval(y2,x2);
y3 = polyfit(x(5:7),y(5:7),2);  y3 = polyval(y3,x3);
xq=[x1,x2,x3]; q = [y1,y2,y3];
%% Plots
plot(x,y,'o-',xx,p,'.-.',xx,sn,'.--',xq,q,'.:',[0,4],[0,0],'k')
title('Several Data Fitting Methods','fontsize',13)
%% Cubic spline imposed boundary data  s'(x1)=a and s'(xn)=b
a = 0;  b = 0;
sb = spline(x,[a,y,b],xx);
plot(x,y,'o-',xx,sn,'.-.',xx,sb,'r.--',[0,4],[0,0],'k')
title('Comparing Natural/Boundary Cubic spline','fontsize',13)
```

4.4 The Best Approximation Problem

Let $f(x)$ be a given continuous function on $a \le x \le b$ and let $p(x)$ be a polynomial. Define the maximum possible error in the approximation of $f(x)$ by $p(x)$ on $[a, b]$:

$$E(p) = \max_{a \le x \le b} |f(x) - p(x)|.$$

For each degree $n > 0$, define the minimax error by the smallest possible value for $E(p)$:

$$\rho_n(f) = \min_{\deg(p) \le n} E(p) = \min_{\deg(p) \le n} \left[\max_{a \le x \le b} |f(x) - p(x)| \right].$$

There exists a unique polynomial p of degree $\le n$ that $\rho_n(f) = E(p)$.

The polynomial is called the minimax polynomial approximataion of degree n denoted by $m_n(x)$.

Accuracy of the Minimax Approximation

$$\rho_n(f) \le \frac{[(b-a)/2]^{n+1}}{(n+1)!\, 2^n} \max_{a \le x \le b} |f^{(n+1)}(x)|.$$

4.4.1 Chebyshev Polynomials

For an integer $n \ge 0$, define the n-th Chebyshev polynomial:

$$T_n(x) = \cos n\theta, \quad \theta = \cos^{-1} x \quad (-1 \le x \le 1). \qquad \text{e.g. } T_0(x) = 1, \quad T_1(x) = x.$$

Using the trigonometric addition formulas, we have the triple recursion relation:

$$T_{n+1}(x) = 2x\, T_n(x) - T_{n-1}(x), \quad n \ge 1. \qquad \text{c.f. } \cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b).$$

The Chebyshev Gauss points is the roots of the Chebyshev polynomial $T_{n+1}(x)$:

$$x_j = \cos\left(\frac{(2j+1)\pi}{2(n+1)}\right), \quad j = 0, 1, \cdots, n. \quad \text{c.f. } T_{n+1}(x) = 2^n \prod_{j=0}^{n}(x - x_j).$$

Note that

$$|T_n(\pm 1)| = 1 \quad \text{and} \quad |T_n(x)| \le 1, \quad -1 \le x \le 1$$

and

$$T_n(x) = 2^{n-1}x^n + \text{lower-degree terms}, \quad n \ge 1.$$

Define a modified version of $T_n(x)$:

$$\tilde{T}_n(x) = \frac{1}{2^{n-1}} T_n(x) = x^n + \text{lower-degree terms}, \quad n \ge 1.$$

Then,

$$|\tilde{T}_n(x)| \le \frac{1}{2^{n-1}}, \quad -1 \le x \le 1, \quad n \ge 1.$$

A polynomial whose highest-degree term has a coefficient of 1 is called a monic polynomial.

Theorem 4.3. *The degree n monic polynomial with the smallest maximum absolute value on $[-1, 1]$ is the modified Chebyshev polynomial $\tilde{T}_n(x)$, and its maximum value on $[-1, 1]$ is $\frac{1}{2^{n-1}}$.*

4.4.2 A Near-Minimax Approximation Method

Let x_0, x_1, x_2, x_3 be the interpolation node points in $[-1, 1]$, and let $c_3(x)$ denote the polynomial of degree ≤ 3 that interpolates $f(x)$ at x_0, x_1, x_2, and x_3. Then, the interpolation error is given by

$$f(x) - c_3(x) = \omega(x)\frac{f^{(4)}(c_x)}{4!}$$

where

$$\omega(x) = (x - x_0)(x - x_1)(x - x_2)(x - x_3).$$

Want to choose the interpolation points x_0, x_1, x_2, x_3 so that

$$\max_{-1 \le x \le 1} |\omega(x)|$$

is made as small as possible.

Note that $\omega(x)$ is a monic polynomial of degree 4:

$$\omega(x) = x^4 + \text{lower-degree terms}$$

The smallest possible value for $\max_{-1 \le x \le 1} |\omega(x)|$ is obtained with

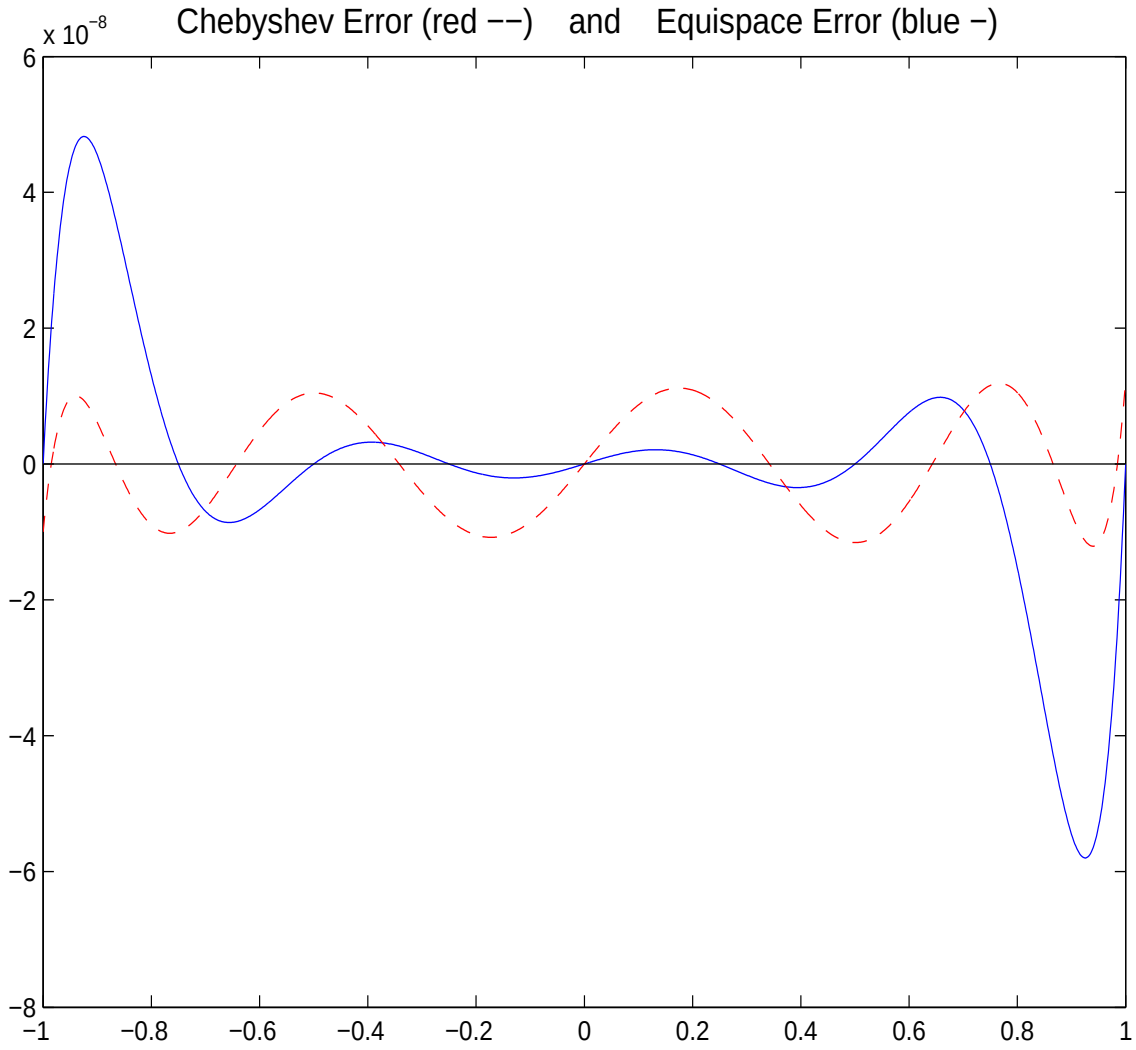
$$\omega(x) = \tilde{T}_4(x) = \frac{T_4(x)}{2^3} = \frac{1}{8}(8x^4 - 8x^2 + 1).$$

and the smallest value is $1/2^3$ in this case.

- The node points are the zeros of $\omega(x)$, and they must therefore be the zeros of $T_4(x)$.

Matlab Code : **(A Near-Minimax Approximation using Chebyshev points)**

```
% cheby_intp.m
% A Near-Minimax Approximation using Chebyshev points
function cheby_intep(n,a,b)
 if nargin==1, a=-1; b=1; end
 CG_pts = inline('cos( (2*[0:n]+1).*pi./(2*n+2) )'); % CG-points function
 ft_f = inline('exp(x)');                            % Example function
 x = a:(b-a)/1000:b;          % Sample points
 ep = [a:(b-a)/n:b];          % Equispaced points
 pn = polyfit(ep,ft_f(ep),n);% Poly Interpolation using Equispace points
 pnx = polyval(pn,x);         % its values
 cg = CG_pts(n); cg = a + (b-a)*(cg+1)./2;  % Chebyshev points on [a,b]
 cn = polyfit(cg,ft_f(cg),n);% Poly. Interpolation using CG-point
 cnx = polyval(cn,x);         % its values
 fx = ft_f(x);
 maxerr_ep = max(abs(fx-pnx));  maxerr_cp = max(abs(fx-cnx));
 figure(1)
   plot(x,fx,x,pnx,x,cnx,'r--',cg,ft_f(cg),'o'); title('Chebyshev Interpolation (red --)')
 figure(2)
   plot(x,fx-pnx,x,fx-cnx,'r--',[a,b],[0,0],'k'); title('Chebyshev Error (red --)')
 fprintf('---- Chebyshev Points and Equispaced Points Interpolation--------\n')
 fprintf('Degree = %g   Max_Error(CP) = %10.4e   Max_Error(EP) = %10.4e \n', ...
          n,maxerr_cp,maxerr_ep);
 fprintf('-------------------------------------------------------------\n')
```


Chebyshev Error (red −−) and Equispace Error (blue −)
x 10^−8
6
4
2
0
−2
−4
−6
−8
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

4.5 Least Squares Approximation

Goal : We want to seek a polynomial $p(x)$ approximating $f(x)$ over $[a,b]$ and minimizing eh L^2-norm error $E(p;f)$.

Let $f(x)$ be a square integrable function on $a \le x \le b$, i.e., $f \in L^2([a,b])$, and let $p(x)$ be a polynomial.

Define the L^2-norm error(or root-square-error) in the approximation of $f(x)$ by $p(x)$ on $[a,b]$:

$$E(p;f) := \left[\int_a^b |f(x)-p(x)|^2 dx\right]^{\frac{1}{2}} \quad \text{denoted by} \quad \|f-p\|.$$

Note that minimizing $E(p;f)$ for different choices of $p(x)$ is equivalent to minimizing $E(p;f)^2$.

Example. Let $f(x) = e^x$ and let $p(x) = a_0 + a_1 x$ over the interval $[-1,1]$. Then, we have

$$\frac{\partial g}{\partial a_0} = -2\int_{-1}^{1}[e^x - a_0 - a_1 x]dx = 0, \quad \frac{\partial g}{\partial a_1} = -2\int_{-1}^{1} x[e^x - a_0 - a_1 x]dx = 0,$$

so that

$$2a_0 = \int_{-1}^{1} e^x dx = e - e^{-1}, \quad \frac{2}{3}a_1 = \int_{-1}^{1} xe^x dx = 2e^{-1},$$

or

$$a_0 = \frac{e-e^{-1}}{2} \approx 1.1752, \quad a_1 = 3e^{-1} \approx 1.1036.$$

Let

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n$$

and let us define

$$g(a_0, a_1, \cdots, a_n) := \int_a^b [f(x) - a_0 - a_1 x - \cdots - a_n x^n]^2 dx$$

and let us seek coefficients $a_0, a_1, \cdots, a_n$ that minimize this integral.

Since the integral $g(a_0, a_1, \cdots, a_n)$ is a quadratic polynomial in the $(n+1)$ variables $a_0, a_1, \cdots, a_n$, a minimizer can be found by invoking the conditions:

$$\frac{\partial g}{\partial a_i} = -2 \int_a^b [f(x) - a_0 - a_1 x - \cdots - a_n x^n] x^i dx = 0, \quad i = 0, 1, \cdots, n.$$

Then, we have the following linear system:

$$\sum_{j=0}^{n} \left(\int_a^b x^{i+j} dx \right) a_j = \int_a^b x^i f(x) dx, \quad i = 0, 1, \cdots, n.$$

With solving the above linear system we can find the least-squares approximation $p(x) = a_0 + a_1 x + \cdots + a_n x^n$.

In this case, if $[a, b] = [0, 1]$, then the coefficient matrix of the resulting linear system is Hilbert matrix of

$$A_{ij} = \frac{1}{i + j + 1},$$

which is known as an ill-conditioned matrix and difficult to solve accurately.

So, we need to introduce a special set of polynomials, e.g., the Legendre polynomial, to get a better approach to minimizing $E(p; f)$.

Define an inner product:

$$(f,g) = \int_a^b f(x)g(x)dx.$$

Let P_j, $j = 0, 1, \cdots, n$ be orthogonal polynomials of degree j, e.g., the Legendre polynomials:

$$(P_i, P_j) = \begin{cases} 0, & i \neq j, \\ \mu_j, & i = j. \end{cases}$$

Let

$$\ell_n(x) = \sum_{j=0}^{n} a_j P_j,$$

and define

$$g(a_0, a_1, \cdots, a_n) = \left(f - \sum_{j=0}^{n} a_j P_j, f - \sum_{j=0}^{n} a_j P_j \right) = (f,f) - \sum_{j=0}^{n} \frac{(f,P_j)^2}{(P_j,P_j)} + \sum_{j=0}^{n} (P_j,P_j) \left[a_j - \frac{(f,P_j)}{(P_j,P_j)} \right]^2.$$

From

$$\frac{\partial g}{\partial a_i} = 2(P_j,P_j) \left[a_j - \frac{(f,P_j)}{(P_j,P_j)} \right] = 0,$$

we have

$$a_j = \frac{(f,P_j)}{(P_j,P_j)}, \quad j = 0, 1, \cdots, n,$$

and then the minimum for this choice of coefficients is

$$g(a_0, a_1, \cdots, a_n) = (f,f) - \sum_{j=0}^{n} \frac{(f,P_j)^2}{(P_j,P_j)}.$$

We call this polynomial

$$\ell_n(x) = \sum_{j=0}^{n} a_j P_j = \sum_{j=0}^{n} \frac{(f, P_j)}{(P_j, P_j)} P_j = \sum_{j=0}^{n} \frac{(f, P_j)}{\mu_j} P_j,$$

the least squares approximation of degree n to $f(x)$ on $[a, b]$.

- Also, we call $\ell_n(x)$ the L^2-orthogonal projection of $f(x)$ upon $\mathbb{P}_n$, the space of polynomials of degree n.
  This means that $\ell_n(x)$ minimizes the L^2-norm error $E(p; f)$ over $\mathbb{P}_n$:

$$E(\ell_n; f) = \min_{p \in \mathbb{P}_n} E(p; f) = \min_{p \in \mathbb{P}_n} ||f - p||.$$

- If P_j are the Legendre polynomials, then the function $\ell_n(x)$ is called the Legendre expansion of degree n for $f(x)$.
  In this case,

$$(P_j, P_j) = \mu_j = \frac{2}{2j+1}.$$

  Note that we can also find the weighted least squares approximation using the following weighted inner product

$$(f, g)_w := \int_a^b f(x) g(x) w(x) dx,$$

  e.g., case of using Chebyshev polynomials.

Matlab Code :

```
-----------------------------------------------------------------
%%%%%%%% Legendre polynomial of degree n %%%%%%%%
function pn = Legen_poly(n);
   pbb = [1]; if n==0, pn=pbb; break; end
   pb = [1 0]; if n==1, pn=pb; break; end
 for i=2:n;
   pn = ( (2*i-1)*[pb,0] - (i-1)*[0, 0, pbb] )/i;
   pbb = pb; pb=pn;
 end
-----------------------------------------------------------------
```

5 Numerical Integration

5.1 Numerical Integration

The definite integral

$$I(f) = \int_a^b f(x)dx$$

is defined in calculus as a limit of what are called Riemann sums.

Numerical integration for $I(f)$ is to find an approximate finite sum:

$$I(f) \approx \sum_{n=0}^{n} w_i f_i$$

with appropriate weights w_i and values f_i.

5.1.1 Trapezoidal Rule

Using the linear polynomial interpolation of $f(x)$ at a and b,

$$P_1(x) = \frac{(b-x)f(a) + (x-a)f(b)}{b-a},$$

we approximate $I(f)$ by the following Trapezoidal rule:

$$I(f) \approx T_1(f) = (b-a)\left[\frac{f(a)+f(b)}{2}\right].$$

If the size of interval $[a,b]$ is big, then we use the following composite Trapezoidal rule: with an integer $n > 1$,

$$h = \frac{b-a}{n}, \quad x_j = a + j * h, \quad j = 0, 1, \cdots, n.$$

Then,

$$I(f) = \int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{n-1}}^{x_n} f(x)dx.$$

$$I(f) \approx T_n(f) = h\left[\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right] = h\sum_{k=1}^{n-1} f(x_k) + \frac{f(x_0)+f(x_n)}{2}h.$$

Error in Trapezoidal rule

$$E_n^T(f) := I(f) - T_n(f) = -\frac{h^2(b-a)}{12}f''(c_n), \quad c_n \in [a,b].$$

5.1.2 Simpson's Rule

To improve on $T_a(f)$, use quadratic interpolation to approximate $f(x)$ on $[a,b]$: with the middle point $c=(a+b)/2$,

$$P_2(x)=\frac{(x-c)(x-b)}{(a-c)(a-b)}f(a)+\frac{(x-b)(x-a)}{(c-b)(c-a)}f(c)+\frac{(x-a)(x-c)}{(b-a)(b-c)}f(b).$$

Let $h=(b-a)/2$. Then

$$\int_a^b \frac{(x-c)(x-b)}{(a-c)(a-b)}dx=\frac{h}{3} \quad \text{and} \quad \int_a^b \frac{(x-b)(x-a)}{(c-b)(c-a)}dx=\frac{4h}{3}\ .$$

Now we have

$$I(f)\approx S_2(f)=\frac{h}{3}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right].$$

If the size of interval $[a,b]$ is big, then we use the following composite Simpson's rule: with an even integer $n>1$,

$$h=\frac{b-a}{n}, \quad x_j=a+j*h, \quad j=0,1,\cdots,n.$$

Then,

$$I(f)=\int_a^b f(x)dx=\int_{x_0}^{x_2} f(x)dx+\int_{x_2}^{x_4} f(x)dx\cdots++\int_{x_{n-2}}^{x_n} f(x)dx.$$

$$I(f)\approx S_n(f)=\frac{h}{3}\Big[f(x_0)+4\Big(f(x_1)+f(x_3)+\cdots+f(x_{n-1})\Big)+2\Big(f(x_2)+2f(x_4)+\cdots+f(x_{n-2})\Big)+f(x_n)\Big].$$

Error in Simpson's rule

$$E_n^S(f):=I(f)-S_n(f)=-\frac{h^4(b-a)}{180}f^{(4)}(c_n), \quad c_n\in[a,b].$$

Matlab Code : **(Simpson's Rule)**

```
--------------------------------------------------------------------------
Q = quad(FUN,A,B) :  Numerically evaluate integral, adaptive Simpson quadrature.

Q = quad(FUN,A,B,TOL) uses an absolute error tolerance of TOL instead of the default,
    which is 1.e-6.

Example:
    FUN can be specified three different ways.

    A string expression involving a single variable:
          Q = quad('1./(x.^3-2*x-5)',0,2);

    An inline object:
          F = inline('1./(x.^3-2*x-5)');
          Q = quad(F,0,2);

    A function handle:
          Q = quad(@myfun,0,2);
          where myfun.m is an M-file:
             function y = myfun(x)
             y = 1./(x.^3-2*x-5);

Also, see quadl, dblquad
--------------------------------------------------------------------------
```

5.1.3 Gaussian Numerical Integration

Let

$$I(f) = \int_{-1}^{1} f(x)dx$$

and consider the following integration formula, so-called the Gaussian Integration formula,

$$I_n(f) = \sum_{j=1}^{n} w_j f(t_j)$$

where the nodes $\{t_1, t_2, \cdots, t_n\}$ and wights $\{w_1, w_2, \cdots, w_n\}$ are chosen that

$$I_n(f) = I(f) \quad \text{for all polynomials of degree } 2n-1.$$

Case $n = 1$: Exact for polynomials $p(x) = 1$ and $f(x) = x$ of degree $2n - 1 = 1$:

From

$$\int_{-1}^{1} 1 \cdot dx = w_1 \cdot 1, \quad \int_{-1}^{1} x dx = w_1 p(t_1) = w_1 \cdot t_1,$$

we have

$$w_1 = 2, \quad t_1 = 0.$$

Case $n = 2$: Exact for polynomials $p(x) = 1, x, x^2, x^3$ of degree $2n - 1 = 3$:

From

$$\int_{-1}^{1} p(x) \cdot dx = I_2(f) = w_1 p(t_1) + w_2 p(t_2),$$

we have

$$w_1 = w_2 = 1, \quad t_1 = -\frac{\sqrt{3}}{3}, \, t_2 = \frac{\sqrt{3}}{3}.$$

For a general interval $[a, b]$, let

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} t_i \quad \text{and} \quad w_i = \frac{b-a}{2}, \quad i = 1, 2, \cdots, n.$$

we have

$$\int_a^b f(x) \cdot dx \approx I_n(f) = \frac{b-a}{2} \sum_{i=1}^{n} w_i f(x_i).$$

Matlab Code : **(Gaussian Quadrature Rule)**

```
Usage:       y = gausquad(a,b,n,f);

 Description: Use the n-point Gauss-Legendre formula to numerically
              integrate the function f(x) over the interval [a,b].

 Inputs:      a = lower limit of integration
              b = upper limit of integration
              n = number of points (1 <= n <= 10)
              f = string containing name of function to be integrated.
                  The form of f is:
                       function y = f(x)
 Examples
     An inline object:
          F = inline('1./(x.^3-2*x-5)');
          Q = gausquad(0,2,5,F);

     A function handle:
          Q = gausquad(0,2,5,myfun);
          where myfun.m is an M-file:
             function y = myfun(x)
             y = 1./(x.^3-2*x-5);

 Outputs:     y = estimate of integral
```

5.1.4 Improper Integration

For the following improper integration

$$\int_{-\infty}^{\infty} f(x)dx = \lim_{X\to\infty} \int_{-X}^{X} f(x)dx,$$

we can use the composite Trapezoidal rule :

```
-------------------------------------------------
In Matlab,
  % First set    h and N  that X = h*N
  x = -N*h:h:N*h;  % node points
  fx = f(x);       % function values at nodes
  In = h*( sum(fx) - ( fx(1)+fx(2*N+1) )/2 );
-------------------------------------------------
```

Example. Find

$$I = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} \exp(-x^2)dx = 1.$$

Consider an another improper integration

$$I = \int_0^1 \frac{\exp(-x^2)}{\sqrt{1-x^2}} dx.$$

We can use linear transformation such as

$$x = \frac{a+b}{2} + \frac{b-a}{2} \tanh z,$$

so that

$$I(f) = \int_a^b f(x)dx = \int_{-\infty}^{\infty} f(x(z)) \frac{dx}{dz} dz = \int_{-\infty}^{\infty} f(x(z)) \frac{b-a}{2\cosh^2 z} dz.$$

```
-----------------------------------------------------------------
In Matlab,
  % First set    h and N  that X = h*N
  z = -N*h:h:N*h;                        % node points
  xz = (a+b)/2 + (b-a)*tanh(z)./2;
  gz = (b-a)/2*f(xz)./cosh(z).^2;      % function values at nodes
  In = h*( sum(gz) - ( gz(1)+gz(2*N+1) )/2 );
-----------------------------------------------------------------
```

6 Introduction to Finite Difference Methods

6.1 FDM for 1D Problems

Let $h = 1/N$ and set

$$t_i = i\,h, \quad I_i = [t_{i-1}, t_i],$$

$$0 = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = 1.$$

Denote by the value of a function $\psi_i = \psi(t_i)$. Also denote by $u_i' = u'(t_i)$ and $u_i'' = u''(t_i)$.

Define a numerical derivative of $u(x)$ with stepsize h:

$$D_h u(t) := \frac{u(t+h) - u(t)}{h} \approx u'(t).$$

From Taylor expansion,

$$u(t_i + h) = u(t_i) + u'(t_i)h + \frac{u''(t_i)}{2}h^2 + \frac{u^{(3)}(t_i)}{3!}h^3 + \frac{u^{(4)}(\xi_i)}{4!}h^4$$
$$u(t_i - h) = u(t_i) - u'(t_i)h + \frac{u''(t_i)}{2}h^2 - \frac{u^{(3)}(t_i)}{3!}h^3 + \frac{u^{(4)}(\xi_i)}{4!}h^4.$$

or

$$
\begin{aligned}
u'(t_i) &= \frac{u(t_i+h)-u(t_i)}{h} - \frac{u''(t_i)}{2}h - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3 \\
u'(t_i) &= \frac{u(t_i)-u(t_i-h)}{h} + \frac{u''(t_i)}{2}h - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3 \\
u'(t_i) &= \frac{u(t_i+h)-u(t_i-h)}{2h} - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3 \\
u''(t_i) &= \frac{u(t_i-h)-2u(t_i)+u(t_i-h)}{h^2} - \frac{u^{(3)}(t_i)}{3!}h^2 + \frac{u^{(4)}(\xi_i)}{4!}h^3.
\end{aligned}
$$

Now, define the following finite differences:

1. The forward finite difference of the first order derivative:

$$u'(t_i) = \frac{u_{i+1} - u_i}{h} + O(h)$$

2. The backward finite difference of the first order derivative:

$$u'(t_i) = \frac{u_i - u_{i-1}}{h} + O(h)$$

3. The central finite difference of the first order derivative:

$$u'(t_i) = \frac{u_{i+1} - u_{i-1}}{2h} + O(h^2)$$

4. The central finite difference of the second order derivative:

$$u''(t_i) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2).$$

Here, one may easily check the error bounds using the Taylor series expansion with the sufficient smooth function u.

Note that the discretization error bound is of the second order if we use the central finite difference formula and one may use more points to get better error bounds.

Consider the following 1-D problem, an initial value problem:

$$\begin{cases} u' + cu = f & \text{in } (0,1), \\ u(0) = \alpha, \end{cases} \tag{1D}$$

Then we have the following approximation scheme using the backward finite difference:

With $u_0 = \alpha$,

$$\frac{u_i - u_{i-1}}{h} + c_i u_i = f_i, \quad i = 1, 2, \cdots, N,$$

or

$$u_i = \left(\frac{1}{1 + c_i\, h}\right) u_{i-1} + \left(\frac{h}{1 + c_i\, h}\right) f_i, \quad i = 1, 2, \cdots, N.$$

Also let us consider the following 1-D problem, two-point boundary value problem:

$$\begin{cases} -u'' + bu' + cu = f & \text{in } (0,1), \\ u(0) = \alpha, \quad u(1) = \beta. \end{cases} \tag{1D}$$

Then we have the following approximation scheme using the central finite differences:

$$\text{With} \quad u_0 = \alpha \quad \text{and} \quad u_N = \beta,$$

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + b_i \frac{u_{i+1} - u_{i-1}}{2h} + c_i u_i = f_i, \quad i = 1, 2, \cdots, N-1$$

or

$$\left(-1 - \frac{b_i\, h}{2}\right) u_{i-1} + (c_i\, h^2 + 2) u_i + \left(-1 + \frac{b_i\, h}{2}\right) u_{i+1} = f_i\, h^2, \quad i = 1, 2, \cdots, N-1.$$

Then, we have the following tridiagonal system when $b = c = 0$:

$$A\,\hat{u} = \hat{f},$$

where

$$A = \text{diag}(-1, 2, -1) := \frac{1}{h^2} * \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix}, \quad \hat{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \quad \text{and} \quad \hat{f} = \begin{bmatrix} f_1 + \alpha/h^2 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-2} \\ f_{N-1} + \beta/h^2 \end{bmatrix}.$$

6.2 FDM for 2D Problems

Let $U = \big(u_{j,i}\big)$ and $F = \big(f_{j,i}\big)$ be matrices consisting of values of function u and f on the grid points (x_j, y_i) such as

$$U = \begin{bmatrix} u_{1,1} & u_{2,1} & \cdots & u_{N,1} \\ u_{1,2} & u_{2,2} & \cdots & u_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1,N} & u_{2,N} & \cdots & u_{N,N} \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} f_{1,1} & f_{2,1} & \cdots & f_{N,1} \\ f_{1,2} & f_{2,2} & \cdots & f_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1,N} & f_{2,N} & \cdots & f_{N,N} \end{bmatrix}$$

To write a linear system in the standard form $A\mathbf{x} = \mathbf{b}$, we need to order the unknown u_{ij} in some way.

For $U = \big(u_{j,i}\big) \in \mathbb{R}^{n,n}$, we define a vector

$$vecU = \big(u_{1,1}, \cdots, u_{1,n}, u_{2,1}, \cdots, v_{2,n}, \cdots, u_{n,1}, \cdots, u_{n,n}\big)^T \in \mathbb{R}^{n^2}$$

by stacking the columns of U on top of each other.

Note that

$$u_{i,j} = vecU\big((i-1)n + j\big).$$

We also need to define a Kronecker product.

For $A \in \mathbb{R}^{p,q}$ and $B \in \mathbb{R}^{r,s}$, define the (right) Kronecker (tensor) product

$$C = A \otimes B := \begin{bmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,q}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,q}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1}B & a_{p,2}B & \cdots & a_{p,q}B \end{bmatrix}.$$

If the system has the following five points stencil

$$\begin{bmatrix} 0 & \beta_3 & 0 \\ \alpha_1 & \alpha_2+\beta_2 & \alpha_3 \\ 0 & \beta_1 & 0 \end{bmatrix},$$

then, without touching the boundary, it can be written by

$$\beta_1\,\hat{u}_{\ell-1} + \beta_2\,\hat{u}_{\ell} + \beta_3\,\hat{u}_{\ell+1} + \alpha_1\,\hat{u}_{\ell-N} + \alpha_2\,\hat{u}_{\ell} + \alpha_3\,\hat{u}_{\ell+N} = \hat{f}_{\ell}$$

where $\hat{f}_\ell = f_{i,j}$ and $\hat{u}_\ell = u_{i,j}$.

Let $J_1 = \mathrm{diag}(\alpha_1, \alpha_2, \alpha_3)$ and $J_2 = \mathrm{diag}(\beta_1, \beta_2, \beta_3)$:

$$J_1 = \mathrm{diag}(\alpha_1, \alpha_2, \alpha_3) := \begin{bmatrix} \alpha_2 & \alpha_3 & 0 & 0 & \cdots & 0 \\ \alpha_1 & \alpha_2 & \alpha_3 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha_1 & \alpha_2 & \alpha_3 \\ 0 & \cdots & \cdots & 0 & \alpha_1 & \alpha_2 \end{bmatrix} \quad \text{and} \quad J_2 = \mathrm{diag}(\beta_1, \beta_2, \beta_3) := \begin{bmatrix} \beta_2 & \beta_3 & 0 & 0 & \cdots & 0 \\ \beta_1 & \beta_2 & \beta_3 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \beta_1 & \beta_2 & \beta_3 \\ 0 & \cdots & \cdots & 0 & \beta_1 & \beta_2 \end{bmatrix}.$$

Then the matrix is given by the block form

$$A = \begin{bmatrix} \alpha_2 I & \alpha_3 I & 0 & 0 & \cdots & 0 \\ \alpha_1 I & \alpha_2 I & \alpha_3 I & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha_1 I & \alpha_2 I & \alpha_3 I \\ 0 & \cdots & \cdots & 0 & \alpha_1 I & \alpha_2 I \end{bmatrix} + \begin{bmatrix} J_2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & J_2 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & J_2 & 0 \\ 0 & \cdots & \cdots & 0 & 0 & J_2 \end{bmatrix}.$$

Furthermore, the matrix A is also given by the Kronecker Products:

$$A = J_1 \otimes I + I \otimes J_2.$$

In Matlab

```
>> J1 = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
>> J2 = bet1*diag(ones(N-1,1),-1) + bet2*diag(ones(N,1)) + bet3*diag(ones(N-1,1),1);
>> A = kron(J1,eye(N)) + kron(eye(N),J2);
```

6.2.1 FDM for Poisson equation

Consider the following Poisson equations:

$$\begin{cases} -\Delta u &= f & \text{in } \Omega = (x_a, x_b) \times (y_a, y_b), \\ u &= 0 & \text{on } \partial\Omega. \end{cases}$$

With $h = \frac{x_b - x_a}{N+1}$ and $k = \frac{y_b - y_a}{N+1}$, we have the following approximation scheme by FDM:

$$-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} = f_{i,j},$$

for $i, j = 1, 2, \cdots, N$.

The five points stencil is given by

$$\begin{bmatrix} 0 & -\frac{1}{k^2} & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + \frac{2}{k^2} & -\frac{1}{h^2} \\ 0 & -\frac{1}{k^2} & 0 \end{bmatrix}.$$

Let J be a tridiagonal matrix:

$$J = \text{diag}(-1, 2, -1) := \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix}$$

and let $J_x = (\frac{1}{h^2})\, J$ and $J_y = (\frac{1}{k^2})\, J$. Then we have the following matrix equation

$$U J_x + J_y U = F.$$

From this equation, we have the following standard matrix form of Poisson Problem:

$$A * vecU = vecF$$

where

$$A = J_x \otimes I + I \otimes J_y.$$

```
-------------------------------------------------------------------------------
In Matlab
  >> h = (xb-xa)/(N+1);    k = (yb-ya)/(N+1);
  >> x = xa+h:h:xb-h;   y = ya+k:k:yb-k;
  >> [x,y] = meshgrid(x,y);       % matrices
  >> f = ft_f(x,y);   f = f(:);

  >> alp1 = -1/h^2; alp2 = 2/h^2; alp3 = -1/h^2;
  >> bet1 = -1/k^2; bet2 = 2/k^2; bet3 = -1/k^2;
  >> Jx = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
  >> Jy = bet1*diag(ones(N-1,1),-1) + bet2*diag(ones(N,1)) + bet3*diag(ones(N-1,1),1);

  >> A = kron(Jx,eye(N)) + kron(eye(N),Jy);
  >> uh = A\f;

  >> Uh = reshape(uh, N, N);
  >> mesh(x, y, Uh)
-------------------------------------------------------------------------------
```

6.2.2 FDM FOR ELLIPTIC EQUATION

Consider the following 2D elliptic problem:

$$\begin{cases} -\Delta u + \mathbf{b}\cdot\nabla u + cu &= f \quad & \text{in } \Omega = (0,1)^2, \\ u &= 0 \quad & \text{on } \partial\Omega, \end{cases} \tag{2D}$$

where $\mathbf{b} = (b^1, b^2)^t$ and c are constant functions.

With $h = k = 1/(N+1)$, we have the following approximate scheme:

$$\begin{aligned} &-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} \\ &+ b^1\frac{u_{i+1,j} - u_{i-1,j}}{2h} + b^2\frac{u_{i,j+1} - u_{i,j-1}}{2k} + c\,u_{i,j} = f_{i,j}, \end{aligned}$$

and then we have the following five points stencil

$$\begin{bmatrix} 0 & -\frac{1}{k^2} + \frac{b^2}{2k} & 0 \\ -\frac{1}{h^2} - \frac{b^1}{2h} & \frac{2}{h^2} + \frac{2}{k^2} + c & -\frac{1}{h^2} + \frac{b^1}{2h} \\ 0 & -\frac{1}{k^2} - \frac{b^2}{2k} & 0 \end{bmatrix}.$$

Let $A = J_x \otimes I + I \otimes J_y$, where

$$J_x = \mathrm{diag}\Big(-\frac{1}{h^2} - \frac{b^1}{2h},\ \frac{2}{h^2} + c,\ -\frac{1}{h^2} + \frac{b^1}{2h}\Big) \quad \text{and} \quad J_y = \mathrm{diag}\Big(-\frac{1}{k^2} - \frac{b^2}{2k},\ \frac{2}{k^2},\ -\frac{1}{k^2} + \frac{b^2}{2k}\Big).$$

Then we have the matrix equation

$$UJ_x + J_yU = F \quad \text{or} \quad A * vecU = vecF.$$

You can also use the following tridiagonal matrices: with $A = J_x \otimes I + I \otimes J_y$, where

$$J_x = \mathrm{diag}\Big(-\frac{1}{h^2} - \frac{b^1}{2h},\ \frac{2}{h^2} + \frac{2}{k^2} + c,\ -\frac{1}{h^2} + \frac{b^1}{2h}\Big) \quad \text{and} \quad J_y = \mathrm{diag}\Big(-\frac{1}{k^2} - \frac{b^2}{2k},\ 0,\ -\frac{1}{k^2} + \frac{b^2}{2k}\Big).$$

In Matlab

```
>> N = input('The number of points used in each axis : N = ');
>> h = 1/(N+1);
>> x = h:h:N*h;    y = x;
>> [x,y] = meshgrid(x,y);        % matrices
>> f = ft_f(x,y);    f = f(:);

>> alp1 = -1/h^2-b1/(2*h); alp2 = 2/h^2+c; alp3 = -1/h^2+b1/(2*h);
>> bet1 = -1/h^2-b2/(2*h); bet2 = 2/h^2; bet3 = -1/h^2+b2/(2*h);
>> Jx = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
>> Jy = bet1*diag(ones(N-1,1),-1) + bet2*diag(ones(N,1)) + bet3*diag(ones(N-1,1),1);

>> A = kron(Jx,eye(N)) + kron(eye(N),Jy);
>> uh = A\f;

>> Uh = reshape(uh, N, N);
>> mesh(x, y, Uh)

  You need to define the data function  'ft_f'.
```

Matlab Code : **(FDM for Elliptic Problem 1)**

```
% Filename fdm_ellip.m
% - Lapace(u) + b1*u_x + b2*u_y + c*u = f  in  O = (0,1)^2
%           u = 0  on boundary

function fdm_ellip(k,b1,b2,c)

if k>9, disp('Too big k !'); return; end
if nargin==1, c=0; b1=0; b2=0;
elseif nargin==2, b2=b1; c=0;
elseif nargin==3, c=0;
end

N = 2^k;  h = 1/(N+1);
x = h:h:N*h; y = x;     % [h 2h 3h ... Nh]
[x,y] = meshgrid(x,y);  % x, y are matrices.

% Right hand side ( Source term )
F = ft_f(x,y,b1,b2,c);  % matrix
F = F(:);       % column vector

% Coefficient matrix
alp1 = -1/h^2-b1/(2*h); alp3 = -1/h^2+b1/(2*h);
bet1 = -1/h^2-b2/(2*h); bet3 = -1/h^2+b2/(2*h);
alp2 = 4/h^2+c;
```

```
Jx = sparse(N,N); Jy = Jx; A = sparse(N^2,N^2);
Jx = alp1*diag(ones(N-1,1),-1) + alp2*diag(ones(N,1)) + alp3*diag(ones(N-1,1),1);
Jy = bet1*diag(ones(N-1,1),-1) + bet3*diag(ones(N-1,1),1);
A = kron(Jx, speye(N)) + kron(speye(N), Jy);
% Approximate solution
uh = A\F;       % vector
% Exact solution
u = ft_u(x,y);  % matrix
% vector -> matrix for mesh
uh = reshape(uh,N,N); % matrix
% Error
e = u-uh; err = h*norm(e(:));
fprintf('--- Elliptic problem with b = (%g,%g), c = %g --- \n',b1,b2,c);
fprintf('N = %g    l2-error = %12.4e \n',N,err);

subplot(131); meshc(x,y,u); title('Exact solution')
subplot(132); meshc(x,y,uh); title('Approximate solution')
subplot(133); meshc(x,y,e); title('Error plot')

%%%%%%%%%%%%%%%%%%%%%%%%%
function u = ft_u(x,y)
  u = (x.^2-x).*sin(pi*y);  % (y.^2-y);

function f = ft_f(x,y,b1,b2,c)
  f = -2*sin(pi*y) + pi^2*(x.^2-x).*sin(pi*y);
  f = f + b1*(2*x-1).*sin(pi*y) + pi*b2*(x.^2-x).*cos(pi*y);
  f = f + c*ft_u(x,y);
```

7 선형 연립방정식

기본적인 선형 연립방정식은 아래와 같다.

$$A\mathbf{x} = \mathbf{b}, \qquad A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

7.1 가우스 소거법 (Gauss Elimination) : 직접 방법 (Direct method)

기본적인 행연산

1) 두 행을 교환한다.
2) 행에 영이 아닌 스칼라를 곱한다.
3) 행에 영이 아닌 스칼라를 곱한 것을 다른 행에 더한다.

가우스 소거법

첨가행렬 $[A\,|\,\mathbf{b}]$에 기본적인 행연산을 하여 $[U\,|\,\tilde{\mathbf{b}}]$ 형태로 바꾸는 방법을 가우스 소거법이라고 한다.

$$[U\,|\,\tilde{\mathbf{b}}] = \left[\begin{array}{cccc|c} u_{1,1} & u_{1,2} & \cdots & u_{1,n} & \tilde{b}_1 \\ 0 & u_{2,2} & \cdots & u_{2,n} & \tilde{b}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & u_{n,n} & \tilde{b}_n \end{array}\right] \quad \text{즉,} \quad A\,\mathbf{x} = \mathbf{b} \xrightarrow{\textit{Gauss Elimination}} U\,\mathbf{x} = \tilde{\mathbf{b}}.$$

단, U는 상부 삼각행렬이다.

역대입법 (back substitution) 이 때, $\mathbf{x}$는 아래의 역대입법으로 찾을 수 있다.

$$
\begin{aligned}
x_n &= \tilde{b}_n / u_{n,n}, \\
x_{n-1} &= (\tilde{b}_{n-1} - u_{n-1,n}\, x_n)/u_{n-1,n-1} \\
x_{n-2} &= (\tilde{b}_{n-2} - u_{n-2,n-1}\, x_{n-1} - u_{n-2,n}\, x_n)/u_{n-2,n-2} \\
&\vdots \\
x_{n-k} &= \left(\tilde{b}_{n-k} - \sum_{j=n-k+1}^{n} u_{n-k,j}\, x_j \right) / u_{n-k,n-k}
\end{aligned}
$$

행렬의 인수분해 (Matrix decomposition)

LU 인수분해 : $A = L\,U$, In Matlab, [L,U] = lu(A)

L은 하부 삼각행렬이며 대각원소는 모두 1이다. U는 상부 삼각행렬이다.

촐리스키 (Cholesky) 인수분해 : $A = L\,L^t$

L은 하부 삼각행렬이며 대각원소는 모두 영이 아니다. L^t는 상부 삼각행렬이 된다.

풀이방법 아래를 풀기 위해 대입법과 역대입법을 사용

$$A\mathbf{x} = \mathbf{b} \;\Rightarrow\; LU\,\mathbf{x} = \mathbf{b} \;\Rightarrow\; \text{Find } \mathbf{y} \text{ satisfying } L\,\mathbf{y} = \mathbf{b} \;\Rightarrow\; \text{Find } \mathbf{x} \text{ satisfying } U\,\mathbf{x} = \mathbf{y}.$$

연산의 횟수 (Number of operations)

가우스 소거법이나 LU 인수분해의 경우 약 $\frac{n^3}{3}$의 연산 횟수가 필요하다.

7.2 반복방법 (Iteration Method)

$A\mathbf{x} = \mathbf{b}$를 풀기 위한 일반적인 반복방법

A의 분리 : $A = N - P,$

이 때, N을 반복행렬이라고 하고, 비특이 행렬이며 풀기 쉽도록 선택

$A\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad N\mathbf{x} = \mathbf{b} + P\mathbf{x}$

초기값 $\mathbf{x}^{(0)}$ 선택

반복실행 : $N\,\mathbf{x}^{(k+1)} = \mathbf{b} + P\,\mathbf{x}^{(k)}, \quad k = 0, 1, 2, \cdots$

위의 반복을 적당한 시점에서 멈춤. 즉, 원하는 오차가 나오는 시점에서 멈춤

오차분석

$N\,(\mathbf{x} - \mathbf{x}^{(k+1)}) = P\,(\mathbf{x} - \mathbf{x}^{(k)})$

$e^{(k+1)}) = N^{-1}\,P\,e^{(k)}), \quad e^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$는 k번째 오차

행렬 노름 $\| N^{-1}\,P \| < 1$을 만족하면 아래와 같이 수렴함

$\|e^{(k)}\| = \|N^{-1}\,P\|^k\,\|e^{(k)}\|$

야코비 방법 (Jacobi method)

N은 A의 대각행렬, $P = N - A$

In Matlab

```
N = diag(diag(A));  P = N - A;
x = zeros(size(b));
for k=1:M
    x = N \ (b + P*x)
end
```

가우스-사이델 방법 (Gauss-Seidel method)

N은 A의 하부 삼각행렬, $P = N - A$

```
> N = tril(A);  P = N - A;
```

8 이산최소자승법

8.1 Discrete Least Squares Approximation (Fitting curve)

Goal : We want to seek a special function $g(x)$ fitting data $\{(x_i, y_i)\}_{i=1}^m$.

Let $\mathcal{B} = \text{span}\big(\phi_1(x), \phi_2(x), \cdots, \phi_n(x)\big)$.

Can you find a function $g(x) = a_1\phi_1(x) + \alpha_2\phi_2(x) + \cdots + a_n\phi_n(x) \in \mathcal{B}$ which satisfies $g(x_i) = y_i$ $(i = 1, 2, \cdots, m)$?

$$g(x_i) = a_1\phi_1(x_i) + \cdots + a_n\phi_n(x_i) = y_i, \qquad \text{or} \qquad A\boldsymbol{\alpha} = Y$$

where

$$A = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_n(x_m) \end{bmatrix}, \quad \boldsymbol{\alpha} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

If $m > n$, then one may not find such a function $g(x) \in \mathcal{B}$ in general.

However, one may find a function $g(x) \in \mathcal{B}$ which minimizes

$$F(g) := \sum_{i=1}^{m} |g(x_i) - y_i|^2.$$

Then, $F(g)$ can be considered as a function in the variables $a_1, a_2, \cdots, a_n$. i.e.,

$$F(a_1, \cdots, a_n) = F(g) = \sum_{i=1}^{m} |g(x_i) - y_i|^2 = \sum_{i=1}^{m} \left| \sum_{k=1}^{n} a_k\phi_k(x_i) - y_i \right|^2.$$

From $\dfrac{\partial F}{\partial a_j} = 0$ for $j = 1, 2, \cdots, n$,

$$\sum_{i=1}^{m} \left(\sum_{k=1}^{n} a_k \phi_k(x_i) - y_i \right) \phi_j(x_i) = 0,$$

or

$$\sum_{k=1}^{n} a_k \left(\sum_{i=1}^{m} \phi_k(x_i)\phi_j(x_i) \right) = \sum_{i=1}^{m} \phi_j(x_i) y_i, \qquad \forall j = 1, \cdots, n.$$

Now, we have the following equation, so-called the normal equation, for the system $A\,\boldsymbol{\alpha} = Y$:

$$A^t A\,\boldsymbol{\alpha} = A^t Y.$$

The solution $\boldsymbol{\alpha} = (A^tA)^{-1}(A^tY)$ to the above normal equation is called the least-squares solution for the system $A\boldsymbol{\alpha} = Y$, and the function $g(x)$ is called the fitting function in the space $\mathcal{B}$ for the data $\{(x_i, y_i)\}_{i=1}^m$.

Example. Find the fitting function given by

$$g(x) = a_1 \sin(\pi x) + a_2 \cos(\pi x) + a_3 \pi x \quad \text{for the data} \quad (1/6, 1),\ (1/4, 0),\ (1/2, -1),\ (3/4, 0).$$

Matlab Code :

```
%% ------------------------------------------------------------
x = [1/6, 1/4, 1/2, 3/4]';   y = [1, 0, -1, 0]';
A = [ sin(pi*x)  cos(pi*x)  pi*x ];
u = A\Y;                      %% u = inv(A^t*A)*(A^t*y)
xx = linspace(1/6,3/4,100);
gxx = u(1)*sin(pi*xx)+u(2)*cos(pi*xx)+u(3)*pi*xx;
plot(x,y,'o',xx,gxx);
%% ------------------------------------------------------------
```

9 고유값 문제

정사각행렬 A에 대하여, 만일 영이 아닌 열벡터 $\mathbf{v} \neq 0$에 대하여

$$A\mathbf{v} = \lambda\mathbf{v}$$

가 성립하면 이 수 λ를 A의 고유값(eigenvalue)이라 한다.
그리고 열벡터 $\mathbf{v}$를 그 해당 고유벡터(eigenvector)라 한다.

대칭행렬의 고유값

행렬 A가 대칭행렬이면 모든 고유값 λ_k는 실수가 되고, 고유벡터 $\mathbf{v}_k$를 열벡터로 가지는 직교행렬 U가 있어 다음을 만족한다.

$$U^t A U = D = diag(\lambda_k), \quad UU^t = I.$$

한편, 대칭행렬 A의 고유값을 λ_k $(k = 1, 2, \cdots, n)$라고 하고 그 해당 고유벡터를 각각 $\mathbf{v}_k$라고 하자.
그러면 모든 벡터 $\mathbf{x} \in \Re^n$은 고유벡터를 성분으로 다음과 같은 일차 결합으로 표현된다.

$$\mathbf{x} = \sum_{k=1}^{n} c_k \mathbf{v}_k, \quad (c_k \in \Re).$$

이 때, $A\mathbf{x}$는 다음과 같이 표현된다.

$$A\mathbf{x} = \sum_{k=1}^{n} \lambda_k c_k \mathbf{v}_k.$$

Matlab Code :

```
A = [ -7 13 -16; 13 -10 13; -16 13 -7 ];
[U,D] = eig(A);
```

A의 최대 고유값

A의 최대 고유값(λ_1)과 그 해당하는 고유벡터($\mathbf{v}_1$)를 찾는 수치방법:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|.$$

1. 초기값 $z^{(0)} = 0$을 선택하자. 무작위로 선택할 수 있음.
2. For $k = 1, 2, \cdots, N$ (N은 적당한 반복횟수)

   Set $w^{(k)} = A\, z^{(k-1)}$

   Set $z^{(k)} = w^{(k)}/\alpha_k, \quad \alpha_k = \max\{\, |w_j^{(k)}|,\ j = 1, 2, \cdots, n\}.$ $w_j^{(k)}$는 $w^{(k)}$의 j번째 성분.

   적당히 큰 k에 대해서 부터 아래와 같이 고유값의 근사값을 구할 수 있다.

   $$\lambda_1^{(k)} = \frac{w_\ell^{(k)}}{z_\ell^{(k-1)}}, \quad \ell \text{은 } z^{(k-1)} \text{의 최대성분이 } \ell \text{번째일 때 선택하고 고정할 수 있음}$$

위에서 찾은 $\lambda_1^{(N)}$과 $z^{(N)}$은 각각 λ_1과 $\mathbf{v}_1$의 상수배로 수렴한다.

$$\lambda_1 - \lambda_1^{(N)} \approx c \left(\frac{\lambda_2}{\lambda_1}\right)^N.$$

10 비선형 연립방정식과 Newton Method

비선형 방정식 $f(x)=0$의 근을 찾기 위한 뉴톤법은 아래와 같다.

1. 이전 근사치 x^n를 준다
2. 곡선 $y=f(x)$에 대하여 $(x^n, f(x^n))$을 지나는 접선(근사직선)의 영점을 찾는다.

$$x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)} = x^n - [f(x^n)]^{-1} f'(x^n)$$

비선형 연립방정식

아래와 같이 비선형 연립방정식을 고려하자.

$$F(x,y) = \begin{cases} f(x,y)=0 \\ g(x,y)=0 \end{cases}$$

그러면, 두 곡면 $z=f(x,y)$와 $z=g(x,y)$의 공통 영점을 찾는 문제가 되는 데,
뉴톤법은 곡면의 접평면(근사평면)의 공통 영점을 찾는 반복법으로 영점을 찾아 가는 방법이다.
먼저 (x^n, y^n)에서 $z=f(x,y)$의 접평면 $z=p(x,y)$는 다음과 같이 주어진다.

$$z = p(x,y) = f(x^n,y^n) + (x-x^n) f_x(x^n,y^n) + (y-y^n) f_y(x^n,y^n)$$

같은 방법으로 $z=g(x,y)$의 접평면 $z=q(x,y)$도 아래와 같이 얻을 수 있다.

$$z = q(x,y) = g(x^n,y^n) + (x-x^n) g_x(x^n,y^n) + (y-y^n) g_y(x^n,y^n)$$

두 접평면의 공통 영점 (x^{n+1}, y^{n+1}) 은 두 접평면의 연립방정식으로 부터 얻을 수 있다.

$$\begin{aligned} f(x^n, y^n) + (x^{n+1} - x^n) f_x(x^n, y^n) + (y^{n+1} - y^n) f_y(x^n, y^n) = 0 \\ g(x^n, y^n) + (x^{n+1} - x^n) g_x(x^n, y^n) + (y^{n+1} - y^n) g_y(x^n, y^n) = 0 \end{aligned} .$$

이를 행렬식으로 표현하면

$$\begin{bmatrix} f_x(x^n, y^n) & f_y(x^n, y^n) \\ g_x(x^n, y^n) & g_y(x^n, y^n) \end{bmatrix} \begin{bmatrix} x^{n+1} - x^n \\ y^{n+1} - y^n \end{bmatrix} = - \begin{bmatrix} f(x^n, y^n) \\ g(x^n, y^n) \end{bmatrix}$$

그러므로 아래와 같은 뉴톤 반복을 찾을 수 있다.

$$\mathbf{x}^{n+1} = \mathbf{x}^n - [F'(\mathbf{x}^n)]^{-1} F(\mathbf{x}^n)$$

여기서

$$\mathbf{x}^n = \begin{bmatrix} x^n \\ y^n \end{bmatrix}, \quad F'(\mathbf{x}^n) = \begin{bmatrix} f_x(x^n, y^n) & f_y(x^n, y^n) \\ g_x(x^n, y^n) & g_y(x^n, y^n) \end{bmatrix}, \quad F(\mathbf{x}^n) = \begin{bmatrix} f(x^n, y^n) \\ g(x^n, y^n) \end{bmatrix}.$$

여기서 $F'(\mathbf{x})$ 를 $F(\mathbf{x})$ 의 야코비행렬 (Jacobian matrix) 라고 한다. 일반적으로

$F'(\mathbf{x}) = (\ \partial_{x_j} f_i(\mathbf{x})\ )$ 로 주어 진다.

11 Initial Value Problem

11.1 General solvability theory

In this section we study the numerical solution for the following initial value problem :

$$(IVP) \qquad \begin{cases} Y'(t) = f(t, Y(t)), & t \geq t_0 \\ Y(t_0) = Y_0. \end{cases}$$

Theorem 11.1. *If $f(t,z)$ and $\partial f(t,z)/\partial z$ be continuous functions of t and z at all points (t,z) in some neighborhood of the initial point (t_0, Y_0). Then there is a unique function $Y(t)$ defined on some interval $[t_0-\alpha, t_0+\alpha]$, satisfying*

$$\begin{cases} Y'(t) = f(t, Y(t)), & t_0 - \alpha \leq t \leq t_0 + \alpha \\ Y(t_0) = Y_0. \end{cases}$$

Note.

1. The number α depends on the initial value problem(IVP).
2. For some equations, e.g., $f(t,z) = \lambda z + b(t)$ where $b(t)$ is continuous, solutions exist for any t, i.e., we can take α to be ∞.
3. For many nonlinear equations, solutions can exist only in bounded intervals.

11.2 Stability

We will generally assume that the solution $Y(t)$ is being sought on a given finite interval $t_0 \le t \le T$.

When small changes in the initial value Y_0 leads to small changes in the solution $Y(t)$, the IVP is said to be stable.

That is, for the solution $Y_\eta(t)$ of the perturbed problem

$$Y_\varepsilon'(t) = f(t, Y_\varepsilon(t)), \quad t_0 \le t \le T, \quad Y_\varepsilon(t_0) = Y_0 + \varepsilon,$$

we have

$$\max_{x_0 \le t \le T} |Y_\varepsilon(t) - Y(t)| \le c\,\varepsilon, \quad \text{some} \;\; c > 0.$$

Example (Stable)**.**

$$Y'(t) = -Y(t) + 1, \quad 0 \le t \le b, \quad Y(0) = 1, \qquad \text{Solution}: \;\; Y(t) \equiv 1,$$

$$Y_\varepsilon'(t) = -Y_\varepsilon(t) + 1, \quad 0 \le t \le b, \quad Y_\varepsilon(0) = 1 + \varepsilon, \qquad \text{Solution}: \;\; Y_\varepsilon(t) \equiv 1 + \varepsilon e^{-t}.$$

$$\Longrightarrow \quad |Y_\varepsilon(t) - Y(t)| = |-\varepsilon e^{-t}| \le |\varepsilon|, \quad t \ge 0.$$

Example (Ill-conditioned when $\lambda b > 0$ is large.)**.**

$$Y'(t) = \lambda[Y(t) - 1], \quad 0 \le t \le b, \quad Y(0) = 1, \qquad \text{Solution}: \;\; Y(t) \equiv 1,$$

$$Y_\varepsilon'(t) = \lambda[Y_\varepsilon(t) - 1], \quad 0 \le t \le b, \quad Y_\varepsilon(0) = 1 + \varepsilon, \qquad \text{Solution}: \;\; Y_\varepsilon(t) \equiv 1 + \varepsilon e^{\lambda t}.$$

$$\Longrightarrow \quad \max_{0 \le t \le b} |Y_\varepsilon(t) - Y(t)| = |\varepsilon|\, e^{\lambda b}, \quad \text{the change in } Y(t) \text{ is quite significant at } t = b.$$

11.3 Direction Fields

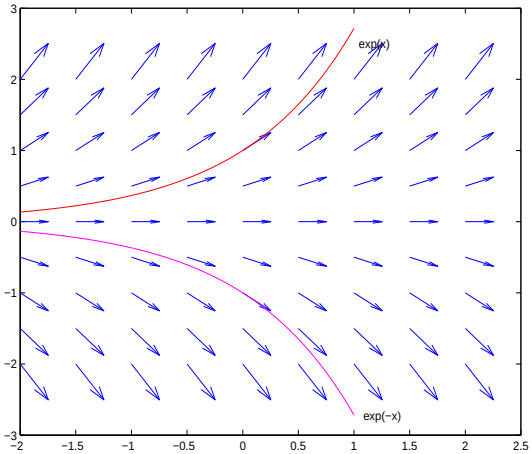
Direction fields are a useful tool to understand the behavior of solutions of a differential equation.

In the graph of a solution of $y' = f(x, y)$, the slope is $f(x, y)$ regardless of the initial value.

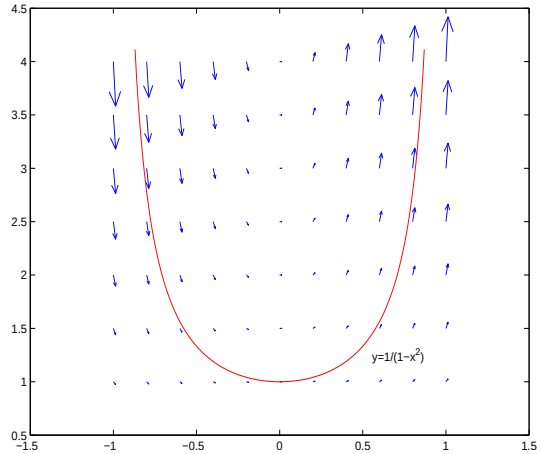
Example. General solution is $y = c\,e^x$ for the D.E. $y' = y$.

Example. General solution is $y = c/(1 - x^2)$ for the D.E. $y' = 2xy^2$.

```
[x,y] = meshgrid(-2:0.5:2, -2:0.5:2);
dx = ones(9);  % one matrix
dy = y; quiver(x,y,dx,dy);
hold on
x = -2:0.01:1;
y1 = exp(x);  y2 = -exp(x);
plot(x,y1,'r',x,y2,'m')
```




```
[x,y] = meshgrid(-1:0.2:1, 1:0.5:4);
dx = ones(7,11);  % one matrix
dy = 2*x.*y.^2; quiver(x,y,dx,dy);
hold on
x = -0.87:0.01:0.87;
y = 1./(1-x.^2);
plot(x,y,'r')
```



11.4 Euler's Method

The simplest, but not efficient numerical method.

The approximate solution will be denoted by $y(t)$ and denote by $y_n = y(t_n)$.

Note that the derivative approximation :

$$Y'(t) \approx \frac{Y(t+h) - Y(t)}{h}.$$

Take the nodes as

$$t_n = t_0 + n * h, \quad n = 0, 1, 2, \cdots, N, \quad \text{with} \quad h = (T - t_0)/N.$$

Applying the derivative approximation to the IVP :

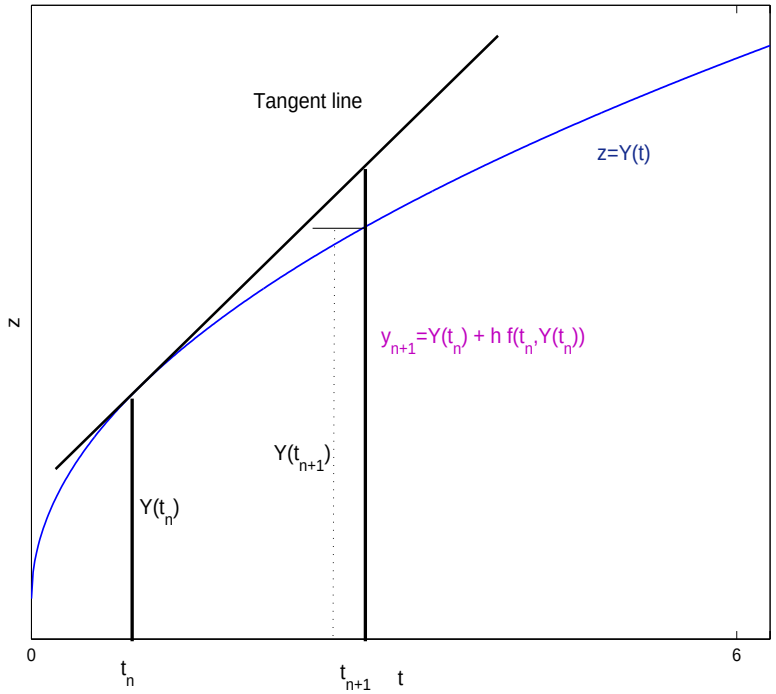
$$Y'(t_n) = f(t_n, Y(t_n)) \quad \text{at} \quad t = t_n,$$

we have

$$\frac{Y(t_{n+1}) - Y(t_n)}{h} \approx f(t_n, Y(t_n)).$$

Euler's method is defined as, with $y_0 = Y_0$

$$y_{n+1} = y_n + h\, f(t_n, y_n), \quad n = 0, 1, 2, \cdots, N-1.$$



Example. Solve $Y'(t) = \dfrac{Y(t) + t^2 - 2}{t+1}, \quad Y(0) = 2$

whose true solution is $Y(t) = t^2 + 2t + 2 - 2(t+1)\log(t+1)$.

11.5 Taylor Methods

Euler's method uses a linear Taylor polynomial approximation :

$$Y(t_{n+1}) \approx Y(t_n) + h\,Y'(t_n) = Y(t_n) + h\,f(t_n, Y(t_n)).$$

Note that the following Chain rule : with $z'(t) = f\big(t, z(t)\big)$

$$z''(t) = \frac{df(t,z(t))}{dt} = \frac{\partial f}{\partial t}\big(t,z(t)\big) + \frac{\partial f}{\partial z}\big(t,z(t)\big)\,z'(t) = f_t\big(t,z(t)\big) + f_z\big(t,z(t)\big)\,f\big(t,z(t)\big).$$

Using a quadratic Taylor polynomial approximation :

$$Y(t_{n+1}) \approx Y(t_n) + h\,Y'(t_n) + \frac{h^2}{2}Y''(t_n),$$

we have the second order Taylor approximation :

$$Y(t_{n+1}) \approx Y(t_n) + h\,f(t_n, Y(t_n)) + \frac{h^2}{2}\left[f_t\big(t_n, Y(t_n)\big) + f_z\big(t_n, Y(t_n)\big)\,f\big(t_n, Y(t_n)\big)\right].$$

The second-order($p = 2$) Taylor method for IVP is given by

$$y_{n+1} = y_n + h\,f(t_n, y_n) + \frac{h^2}{2}\left[f_t\big(t_n, y_n\big) + f_z\big(t_n, y_n\big)\,f\big(t_n, y_n\big)\right], \quad \text{Error : } |Y(t_n) - y(t_n)| = O\big(h^{p+1}\big).$$

Example. Solve

$$Y'(t) = -Y(t) + 2\cos t, \quad Y(0) = 1 \quad \text{whose true solution is } \; Y(t) = \sin t + \cos t.$$

Since $Y''(t) = -Y'(t) - 2\sin t = Y(t) - 2\cos t - 2\sin t,$

Taylor Scheme : $y_{n+1} = y_n + h[-y_n + 2\cos t_n] + \frac{h^2}{2}\left[y_n - 2\cos t_n - 2\sin t_n\right], \quad n \geq 0.$

11.6 Runge-Kutta Methods

The Taylor method needs to calculate the higher-order derivatives.

The Runge-Kutta methods evaluate $f(t, z)$ at more points,

while attempting to equal the accuracy of the Taylor approximation.

RK methods are among the most popular methods for solving IVP.

The Runge-Kutta methods of order 2 : RK2

The general form

$$y_{n+1} = y_n + h\, F(t_n, y_n; h), \quad n \geq 0, \quad y_0 = Y_0$$

where $F(t_n, y_n; h)$ can be thought of as some kind of "average slope" of the solution on the interval $[t_n, t_{n+1}]$.

Choose

$$F(t, y; h) = \gamma_1\, f(t, y) + \gamma_2\, f\big(t + \alpha h, y + \beta h f(t, y)\big).$$

Determine the constants $\alpha, \beta, \gamma_1, \gamma_2$ so that the following truncation error will be $O(h^3)$,

just as with the Taylor method of order 2 :

$$T_{n+1} = Y(t_{n+1}) - [Y(t_n) + h\, F(t_n, Y(t_n); h)].$$

Note that the first-order Taylor expansion for two-variable function $f\big(t, y\big)$:

$$f\big(t + A, y + B\big) = f\big(t, y\big) + A\, f_t(t, y) + B\, f_z(t, y) + O(A^2 + B^2).$$

Hence, we have

$$f\big(t + \alpha h, y + \beta h f(t, y)\big) = f\big(t, y\big) + \alpha\, h\, f_t(t, y) + \beta\, h\, f_z\big(t, y\big)\, f(t, y) + O(h^2).$$

Using $Y'' = f_t + f_z\, f$ yields

$$\begin{aligned} Y(t+h) &= Y + h\,Y' + \frac{h^2}{2}Y'' + O(h^3) \\ &= Y + h\,f + \frac{h^2}{2}\big(f_t + f_z\, f\big) + O(h^3). \end{aligned}$$

Then

$$\begin{aligned} & Y(t+h) - \big[Y(t) + h\,F(t, Y(t); h)\big] \\ &= \big[Y + h\,f + \frac{h^2}{2}\big(f_t + f_z\, f\big)\big] - \big[Y + h\,\gamma_1\, f + h\,\gamma_2\, f\big(t + \alpha h, Y + \beta h f(t, Y)\big)\big] + O(h^3) \\ &= \big[Y + h\,f + \frac{h^2}{2}\big(f_t + f_z\, f\big)\big] - \big[Y + h\,\gamma_1\, f + h\,\gamma_2\,\big(f + \alpha\, h\, f_t + \beta\, h\, f_z\, f\big)\big] + O(h^3) \\ &= h\,(1 - \gamma_1 - \gamma_2)\, f + \frac{h^2}{2}\big[(1 - 2\alpha\gamma_2) f_t + (1 - 2\beta\gamma_2) f_z\, f\big] + O(h^3). \end{aligned}$$

So the coefficients must satisfy the system

$$1 - \gamma_1 - \gamma_2 = 0, \quad 1 - 2\alpha\gamma_2 = 0, \quad 1 - 2\beta\gamma_2 = 0.$$

Therefore,

$$\gamma_2 \neq 0, \quad \gamma_1 = 1 - \gamma_2, \quad \alpha = \beta = \frac{1}{2\gamma_2}.$$

The three favorite choices are $\gamma_2 = \dfrac{1}{2},\ \dfrac{3}{4},\ 1$.

With $\gamma_2 = 1/2$, we have the following RK2 method :

$$y_{n+1} = y_n + \frac{h}{2}\big[f(t_n, y_n) + f\big(t_n + h, y_n + h f(t_n, y_n)\big)\big].$$

```
-----------------------------------
 Runge-Kutta Method of order 2
-----------------------------------
   v1 = f( t(n), y(n) );
   v2 = f( t(n)+h, y(n)+h*v1 );
 y(n+1) = y(n) + (h/2)*( v1 + v2 );
-----------------------------------
```

The Runge-Kutta methods of order 4 : RK4

The truncation error in this method is $O(h^5)$:

$$
\begin{aligned}
v_1 &= f(t_n, y_n) \\
v_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} v_1\right) \\
v_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} v_2\right) \\
v_4 &= f(t_n + h, y_n = h\, v_3) \\
y_{n+1} &= y_n + \frac{h}{6}\big[v_1 + 2v_2 + 2v_3 + v_4\big].
\end{aligned}
$$

Example. Using RK2 and RK4 with $h = 0.1$ and 0.05, solve

$$Y'(t) = \frac{1}{4} Y(t)\left(1 - \frac{1}{20} Y(t)\right) \quad 0 \le t \le 20, \quad Y(0) = 1 \quad \text{whose true solution is} \quad Y(t) = \frac{20}{1 + 19\, e^{-x/4}}.$$

```
-----------------------------------------------
   h     t(n)      Y(n)      y(n)       |Y(n)-y(n)|
-----------------------------------------------
 0.1     2.0
         4.0
          .
-----------------------------------------------
```

11.7 The 2nd-order initial value problem

Consider

$$(IVP) \qquad \begin{cases} y''(t) + a(t)y'(t) = f(t, y(t)), & t \geq t_0 \\ y(t_0) = \alpha, \quad y'(t_0) = \beta. \end{cases}$$

Convert to an equivalent system of 1st-order IVP

Set $\quad y_1(t) = y(t), \quad y_2(t) = y'(t)$

$$(IVP) \qquad \begin{cases} y_1'(t) = y_2(t), & y_1(t_0) = \alpha \\ y_2'(t) = f(t, y_1(t)) - a(t)y_2(t), & y_2(t_0) = \beta \end{cases} \quad \Leftrightarrow \quad Y'(t) = F(t, Y(t)), \quad Y(t_0) = Y_0$$

```
-------------------------------------------------------------------
 Runge-Kutta Method of order 2 : V1, V2, Y(n) are all vector values
-------------------------------------------------------------------
  V1 = f( t(n), Y(n) );   V2 = f( t(n)+h, Y(n)+h*V1 );
  Y(n+1) = Y(n) + (h/2)*( V1 + V2 );
```

Example. Using RK2 and RK4 with $h = 0.1$ and 0.05, solve

$$y''(t) + 11y'(t) + 10y(t) = 10, \quad y(0) = 0,\ y'(0) = 0 \quad \text{whose true solution is} \quad y(t) = 1 - \frac{10}{9}e^{-t} + \frac{1}{9}e^{-10t}.$$