Newton interpolating polynomial

x:	0	1	3	4	7
y:	2	–3	0	1	–2

$$
\begin{array}{cccccc}
0 & \boxed{2} & & & & \\
 & & \boxed{-5} & & & \\
1 & -3 & & \boxed{\tfrac{13}{6}} & & \\
 & & \tfrac{3}{2} & & \boxed{-\tfrac{7}{12}} & \\
3 & 0 & & -\tfrac{1}{6} & & \boxed{\tfrac{19}{252}} \\
 & & 1 & & -\tfrac{1}{18} & \\
4 & 1 & & -\tfrac{1}{2} & & \\
 & & -1 & & & \\
7 & -2 & & & &
\end{array}
$$

$$
2-5x+\frac{13}{6}x(x-1)-\frac{7}{12}x(x-1)(x-3)
$$
$$
+\frac{19}{252}x(x-1)(x-3)(x-4)
$$

Substitute and verify. Plot with the original data.

Lagrange interpolating polynomial

We can use our Maple program, correspondingly modified.

$$
\frac{(x-1)(x-3)(x-4)(x-7)}{42}+\frac{x(x-3)(x-4)(x-7)}{12}
$$
$$
-\frac{x(x-1)(x-3)(x-7)}{36}-\frac{x(x-1)(x-3)(x-4)}{252}
$$

Substitute and verify. Plot, together with the original data.

Least-sqare fit of a polynomial to discrete data

Minimize by solving normal equations

$$
\begin{bmatrix}
n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \dots & \sum_{i=1}^{n} x_i^k \\
\sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \dots & \sum_{i=1}^{n} x_i^{k+1} \\
\sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \dots & \sum_{i=1}^{n} x_i^{k+2} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \sum_{i=1}^{n} x_i^{k+2} & \dots & \sum_{i=1}^{n} x_i^{2k}
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}
=
\begin{bmatrix}
\sum_{i=1}^{n} y_i \\
\sum_{i=1}^{n} x_i y_i \\
\sum_{i=1}^{n} x_i^2 y_i \\
\vdots \\
\sum_{i=1}^{n} x_i^k y_i
\end{bmatrix}
$$

Be able to compute residues, sum of squares residues, typical error, etc. Full pivoting must be used when switching to decimal.

Same with weights

$$\sum_{i=1}^{n} w_i \left[y_i - (a_0 + a_1 x_i + a_2 x_i^2 ... + a_k x_i^k)\right]^2$$

$$\begin{bmatrix} \sum\limits_{i=1}^{n} w_i & \sum\limits_{i=1}^{n} w_i x_i & \dots & \sum\limits_{i=1}^{n} w_i x_i^k \\ \sum\limits_{i=1}^{n} w_i x_i & \sum\limits_{i=1}^{n} w_i x_i^2 & \dots & \sum\limits_{i=1}^{n} w_i x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum\limits_{i=1}^{n} w_i x_i^k & \sum\limits_{i=1}^{n} w_i x_i^{k+1} & \dots & \sum\limits_{i=1}^{n} w_i x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum\limits_{i=1}^{n} w_i y_i \\ \sum\limits_{i=1}^{n} w_i x_i y_i \\ \vdots \\ \sum\limits_{i=1}^{n} w_i x_i^k y_i \end{bmatrix}$$

Least-sqare fit of a linear model to discrete data

Minimize

$$\sum_{i=1}^{n} \left[y_i - (a_1 f_1(x_i) + a_2 f_2(x_i) ... + a_k f_k(x_i))\right]^2$$

by solving normal equations

$$\begin{bmatrix} \sum\limits_{i=1}^{n} f_1(x_i)^2 & \sum\limits_{i=1}^{n} f_1(x_i) f_2(x_i) & \dots & \sum\limits_{i=1}^{n} f_1(x_i) f_k(x_i) \\ \sum\limits_{i=1}^{n} f_2(x_i) f_1(x_i) & \sum\limits_{i=1}^{n} f_2(x_i)^2 & \dots & \sum\limits_{i=1}^{n} f_2(x_i) f_k(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \sum\limits_{i=1}^{n} f_k(x_i) f_1(x_i) & \sum\limits_{i=1}^{n} f_k(x_i) f_2(x_i) & \dots & \sum\limits_{i=1}^{n} f_k(x_i)^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum\limits_{i=1}^{n} y_i f_1(x_i) \\ \sum\limits_{i=1}^{n} y_i f_2(x_i)) \\ \vdots \\ \sum\limits_{i=1}^{n} y_i f_k(x_i) \end{bmatrix}$$

Be able to compute residues, sum of squares residues, typical error, etc. Full pivoting required.

Gaussian elimination with backward substitution

Include full pivoting when the entries are decimal numbers.

Gram-Schmidt orthogonalization

Build the first few polynomials and the corresponding αs.

Least-square 'continuous' fit of a function by a polynomial

Minimize

$$\int_A^B \left[f(x) - (a_0 + a_1 x + a_2 x^2 ... + a_k x^k)\right]^2$$

by:

- In $f(x)$, replace x by $\frac{A+B}{2}+\frac{B-A}{2}X$.
- Compute
$$c_i=\frac{\int_{-1}^{1} f(X)\cdot\phi_i(X)dX}{\alpha_i} \qquad i=0,1,2,...k$$
where ϕ_i are the Legendre polynomials.
- In
$$c_0\phi_0(X)+c_1\phi_1(X)+c_2\phi_2(X)...+c_k\phi_k(X)$$
replace X by $\frac{2x-(A+B)}{B-A}$ and expand.

If we want to use Chebyshev polynomials instead of Legendre, the only thing which changes is
$$c_i=\frac{\int_{-1}^{1}\frac{f(X)\cdot\phi_i(X)}{\sqrt{1-X^2}}dX}{\alpha_i} \qquad i=0,1,2,...k$$
where ϕ_i are now the Chebyshev polynomials.

Compute typical and the largest error, plot, etc.

Composite trapezoidal rule

To approximate
$$\int_A^B y(x)dx$$
we choose a value of n and then compute
$$\frac{y(A)+2y(A+h)+2y(A+2h)+...+2y(B-h)+y(B)}{2n}\cdot(B-A)$$
where $h=\frac{B-A}{n}$. The main error is proportional to h^2 and can be removed by Romberg's algorithm (if you double n, use 4, if you triple it, use 9, etc.).

The *remaining* error is h^4-proportional, and can be removed by a second stage of Romberg's (when n doubles, use 16, when it triples use 81, etc.).

The remaining error is h^6-proportional, and can be removed by one more stage of Romberg's (when n doubles use 64, etc.).

Composite Simpson's rule

To approximate
$$\int_A^B y(x)dx$$
we choose a value of n (must be even) and then compute
$$\frac{y(A)+4y(A+h)+2y(A+2h)+...+4y(B-h)+y(B)}{3n}\cdot(B-A)$$
where $h=\frac{B-A}{n}$. The main error is proportional to h^4 and can be removed by Romberg's algorithm (if you double n, use 16, if you triple it, use 81, etc.).

The *remaining* error is h^6-proportional, and can be removed by a second stage of Romberg's (when n doubles, use 64, when it triples use 729, etc.).

n-point Gaussian formula

To approximate

$$\int_A^B y(x)dx$$

we now compute

$$(B-A)\sum_{i=1}^{n} c_i \cdot y\left(\tfrac{A+B+X_i(B-A)}{2}\right)$$

where X_i are the roots of n-degree Legendre polynomial, and c_i are numerical coefficients (both the X_i and the corresponding c_i are tabulated in your textbook, we also know how to compute them on our own).

We should be able to also apply it as a *composite* formula.

Designing own n-point formula

to approximate

$$\int_A^B w(x) \cdot y(x)dx$$

where A and B are now *specific* numbers and $w(x)$ is a *specific* weight function (may be $\equiv 1$), and x_1, x_2, ... x_n are given nodes (from the A to B interval).

- Using the nodes, find the corresponding Lagrange interpolating polynomial, say $p_n(x)$, to go through $[x_1, y(x_1)]$, $[x_2, y(x_2)]$, ...$[x_n, y(x_2)]$.
- Evaluate

  $$\int_A^B w(x) \cdot p_n(x)dx = c_1 \cdot y(x_1) + c_2 \cdot y(x_2) + ... + c_n \cdot y(x_n)$$

  This is your rule of integration.

Be able to apply it to a specific case of $y(x)$.

If we want to derive a **Gaussian** rule for the above integral, the nodes would not then be given; instead, we have to find the corresponding *orthogonal polynomials* (Gram-Schmidt, using the given interval and weight function), find the roots of the n-degree polynomial, and use *them* as the n nodes. The rest of the procedure is the same.

Designing n-point formulas for k^{th} derivative

The nodes are given as .... $x_0 - 2h$, $x_0 - h$, x_0, $x_0 + h$, $x_0 + 2h$, ...

Using these nodes, we fit the corresponding Lagrange interpolating polynomial, differentiate it k times, and then substitute x_0 for x (and simplify). Be able to apply the resulting formula to a specific function. Also, be able to find its error expansion, based on which perform Richardson extrapolation.

Also, remember that the most basic such formulas are

$$y'(x_0) \simeq \frac{y(x_0+h) - y(x_0-h)}{2h}$$

and

$$y''(x_0) \simeq \frac{y(x_0+h) - 2y(x_0) + y(x_0-h)}{h^2}$$

LU decomposition

Be able to solve a *tri-diagonal* system of linear equations

$$\mathbb{A}\mathbf{x} = \mathbf{r}$$

via LU decomposition of $\mathbb{A}$.

Solving n non-linear equations for n unknowns

The key formula (one iteration) is

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \left[\left. \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right|_{\mathbf{x}_i} \right]^{-1} \mathbf{F}(\mathbf{x}_i)$$

where $\mathbf{F}(\mathbf{x}_i)$ is the vector whose components are the n equations, and $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is the corresponding Jacobian (always evaluated with the most current values of $\mathbf{x}$). Understand the concept of *quadratic convergence.*

Second-order ODE - boundary-value problem

The differential equation may be either linear

$$y'' + p(x) \cdot y' + q(x) \cdot y = r(x)$$

or non-linear

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

We divide the original interval into n subintervals, and find the values of y only at these points. We can then double (or triple) the value of n (once or twice), and improve the solution by Richardson extrapolation.

Solving (sets of) ODE by Runge-Kutta - initial-value problem

The equation looks as follows

$$\dot{y} = f(t, y)$$

where y (and correspondingly f) can have more than one component (when dealing with a *set* of equations).

The fourth-order (referring to the time-step error) Runge-Kutta works like this (one step):

Compute

$$\begin{aligned} k_1 &= hf(t,y) \\ k_2 &= hf(t+\tfrac{h}{2}, y+\tfrac{k_1}{2}) \\ k_3 &= hf(t+\tfrac{h}{2}, y+\tfrac{k_2}{2}) \\ k_4 &= hf(t+h, y+k_3) \end{aligned}$$

then advance t by h and y by

$$h \cdot \frac{k_1+2k_2+2k_3+k_4}{6}$$

Should be able to deal with higher-order ODE by treating y, $\dot{y}$, $\ddot{y}$, ...(one has to go to one less than the equation's order) as individual components of the solution.

Be able to estimate the error of your solution (by backtracking), and realize that going twice as slow (in terms of h) will improve the accuracy 16 fold (going ten times as fast will yield 4 extra digits of accuracy).

Least-square fit of a function by trigonometric polynomial

First, introduce a new scale X by

$$x \to \frac{B-A}{2\pi}\cdot X + \frac{A+B}{2}$$

then compute

$$\begin{aligned} a_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(X)\cos(kX)\ dX \\ b_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(X)\sin(kX)\ dX \end{aligned}$$

(as many as desired). The resulting trigonometric polynomial is:

$$\begin{aligned} &\frac{a_0}{2} + a_1\cos X + b_1 \sin X \\ &+a_2\cos 2X + b_2\sin 2X \\ &+a_3\cos 3X + b_3\sin 3X + ... \end{aligned}$$

where X needs to be replaced by

$$\frac{2x-(A+B)}{B-A}\cdot\pi$$

Be able to deal with functions given in a piecewise manner.

Least-square fit of discrete data by a trigonometric polynomial

Fit the y_i values ($2m$ of them) by computing

$$\begin{aligned} a_k &= \frac{1}{m}\sum_{i=0}^{2m-1} y_i \cos(kX_i) \qquad k = 0,1,2,....n \\ b_k &= \frac{1}{m}\sum_{i=0}^{2m-1} y_i \sin(kX_i) \qquad k = 1,2,.....n-1 \end{aligned}$$

where

$$X_i = -\pi + \frac{\pi}{m}\cdot i \qquad i = 0,1,2,....2m-1$$

construct your polynomial by

$$\begin{aligned} &\frac{a_0}{2} + a_1\cos X + b_1 \sin X \\ &+a_2\cos 2X + b_2\sin 2X \\ &+a_3\cos 3X + b_3 \sin 3X + ... \\ &\qquad\qquad\vdots \\ &+a_{n-1}\cos(n-1)X + b_{n-1}\sin(n-1)X \\ &+a_n \cos nX \end{aligned}$$

then replace X by

$$\frac{x-A}{B-A}\cdot\pi\left(2-\frac{1}{m}\right)-\pi$$

where A and B are the first and last values (respectively) of x in the given table (the x's must follow algebraic progression).

To construct **interpolating** trigonometric polynomial, we simply take $n = m$ (it would be pointless to go any higher), and change the last term to

$$+\frac{a_m}{2}\cos mX$$

The resulting fit must be perfect (no residuals).