

CHAPTER 4

NUMARICAL METHODS

INTERPOLATION AND CURVE FITTING

4.1 INTRODUCTION

In the present and the next three chapters, we shall be dealing with several numerical methods for solving problems which are very common in science and engineering. Problems such as $\frac{dy}{dx} = e^{2x}$, $y = ?$ are exactly soluble in the solution is $y = e^{2x/2}/2 + c$ ($c = \text{constant}$) and y can be evaluated as accurately as desired. For a very large number of realistic problems, e.g. $\frac{dy}{dx} = e^{x^2}$, (differential equation) or $f(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$ (a polynomial equation of degree 5, which has five roots, i.e. five values of x , say x_i ; $i = 1, 5$ for which $f(x_i) = 0$) do not have exact solutions. For these problems, we need to construct numerical methods to obtain solutions. Numerical methods, when successful, yield accurate (not exact) solutions. The subject of numerical analysis analyses the accuracy of these numerical methods. In our four chapters, we will describe elementary algorithms for the commonly used procedures for numerical methods such as interpolation, line/curve fitting, matrix inversion, roots of polynomials, integration, differential equations, integral transforms and stochastic methods. Numerical methods use different approximation schemes, iterative methods, and recursive methods and so on. If we get a solution to a desired accuracy, the method is said to have converged to the answer (of course up to that accuracy). In complex or even some simple problems, converge is not guaranteed! Thus, similar to organic synthesis, numerical analysis and computer programming is a science as well as an art. Let us begin with error analysis.

4.2 ERROR ANALYSIS

Experimental data is accurate only up to some decimal points. On a centimeter scale, we cannot measure lengths to an accuracy of better than 0.1 mm. Similarly, a burette reading cannot be more accurate than 0.01 ml. When we do numerical operations with these data, errors propagate through the arithmetic/numerical operations and the final result has a meaning only up to some accuracy. We will illustrate the error propagation in a few elementary operations. Let us see what the errors in a few common operations are. Represent numbers as $F_x 10^{e_x}$ where F_x is the floating point part of a number and e_x is the exponent. Let $x = F_x 10^{e_x}$ and $y = F_y 10^{e_y}$. If the error in x is $\epsilon_x = x^1 - x$ (where x^1 is the true value of x ; many times the true values are not known, so that we have only some idea of the maximum error value), and that in y is ϵ_y , then the error in the sum of x and y is $\epsilon_x + \epsilon_y + \alpha$ where α is the round off error. Resulting in rounding of the sum $x + y$ to certain maximum digits specified on the computer. Error in $x + y = \epsilon_x + \epsilon_y + \alpha$ where α is the round off error. Error in $x - y = \epsilon_x - \epsilon_y + \sigma$ where σ is the round off error. Error in x/y is better calculated in terms of the relative errors. Let the relative error in x , $r_x = \epsilon_x / x$ similarly $r_y = \epsilon_y / y$. Then $r_{xy} = \epsilon_{xy} / (x + y) = r_x + r_y + \mu$ and the relative error in x/y is $r_x - r_y + \delta$.

Problem: $\epsilon_{xy} = (x^1 y^1 - xy) = (x + \epsilon_x)(y + \epsilon_y) = xy + \epsilon_x y + \epsilon_y x + \epsilon_x \epsilon_y - xy \approx \epsilon_x y + \epsilon_y x$, ignoring $\epsilon_x \epsilon_y$ which is smaller than ϵ_x or ϵ_y

E.g. if $x = 0.4836 \times 10^3$ and $y = 0.5123 \times 10^2$, $xy = 0.24774828 \times 10^5$

$$= 0.2477 \times 10^5 + 0.4828 \times 10^{5-4}$$

$$= F \times 10^{e_x} + f \times 10^{e_x - d}$$

Where d is the number of digits after the decimal point. $f \times 10^{e_2-d}$ is the error and if $|f| < 0.5$ the maximum error is $|f| \times 10^{e_2-d}$ and if $|f| > 0.5$, then maximum error is $|1 - f| \times 10^{e_2-d}$.

These are round off errors. We can thus see that the maximum relative error is $(0.5 \times 10^{e_2-d}) / 0.1 \times 10^{e_2}$ or 5×10^d . When the number of operations increases, errors too increase. If 10^6 numerical operations are involved with all the numbers in a problem (such as, say 10^6 steps of a molecular dynamics trajectory, it is safer to use decimal numbers in double precision (or real*8) where each number has a representation of 16 digits after the decimal point. Only then can we be reasonably sure that the error is beyond the 8th digit.

In reality, errors do not always increase! There is a statistical analysis that treats each error as a random variable. Local round off errors may be treated as uniformly or normally (Gaussian function) distributed and an upper bound to the error in the final result can be arrived at.

Problems, (test problems) – 3 in number

4.3 INTERPOLATION

In experimental measurements, there is bound to be some error (human error, limitation of the precision of the measuring device). A simple tabular form of data makes it very hard to perform any operations with the data. If we can represent the data as a great bonus, have estimates of the function for all the points in the range where measurements were made without performing any additional experiments. Don't fear that we are getting something for nothing! We only have a well educated guess of the results of experiments at intermediate points. Actual experiment is the final proof! There are two ways of finding suitable functions. One is referred as interpolation wherein the function coincides with the experimental data at all the measured values of $f(x_i)$.

Consider the four data points shown in Fig. 4.1

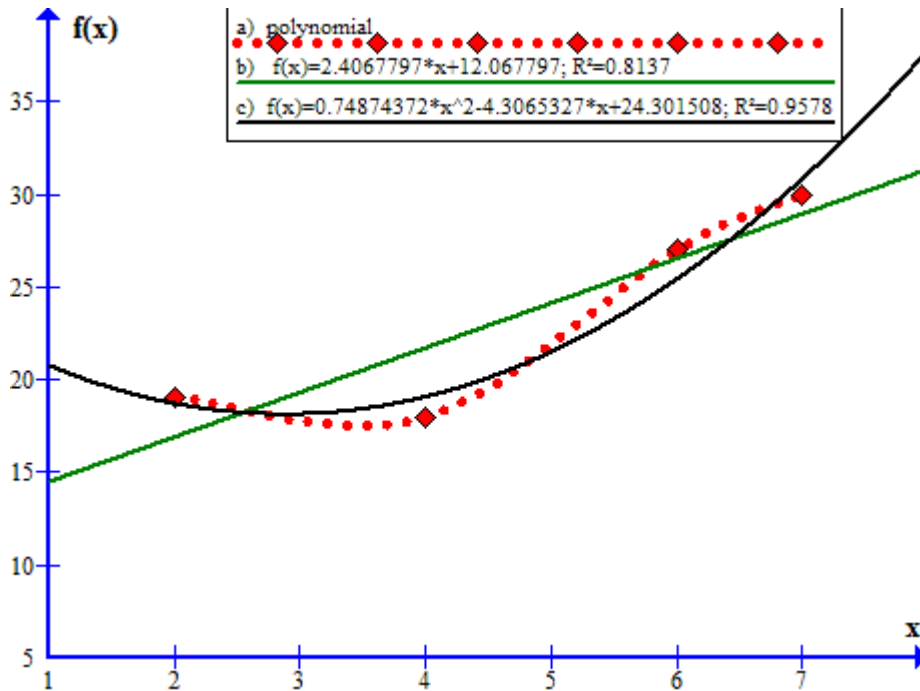


Fig. 4.1 the data points are shown as diamonds. The interpolating polynomial is shown in red color and it passes through all the points. The best fitting straight line is shown in green. The best quadratic fit is shown in black. Note that the best fitting curves do not have to pass through all the data points. The equations for the fitting curve are given in the rectangular box which is included in the figure. The above graph has been plotted using the software setup graph 4.3 which can be downloaded from the site <http://www.padowan.dk/graph/Download.php>

The interpolating polynomial is shown as curve (a) in Fig 4.1. The second option is good fit through all the points. Both straight line (b) and quadratic (c) curves are fits to the data. A few error bars are indicated. If the error bars at all the points are as large as the error in the second point, then both curves b and c are good. If the errors are very small, then interpolation is a better choice. Even then getting a good fitting function near enough to the points may throw good light on the physical features of the problem on hand. There are many ways of finding functions to interpolate at points other than those at which the values of the function are available. Suppose we know the values of the function

$f(x), f(x_0), f(x_1), \dots, f(x_n)$ at the values of $x, x_0, x_1, x_2, \dots, x_n$. We need values of the function at $x_0 < x < x_1, x_1 < x < x_2, \dots$ and $x_{n-1} < x < x_n$. Then the Lagrange interpolating polynomial is defined by

$$P_n(x) = \sum_{k=0}^n f(x_k) L_k(x) / L_k(x_k) \quad - (4.1)$$

Where $L_k(x) = (x - x_0)(x - x_1) \dots (x - x_{k-1})(x - x_{k+1}) \dots (x - x_n)$

And $L_k(x_k) = (x_k - x_0)(x_k - x_1) \dots (x_k - x_{k-1})(x_k - x_{k+1}) \dots (x_k - x_n) \quad - (4.2)$

Note that for the k^{th} function, the term $x - x_k$ or $x_k - x_k$ is absent in 4.2. It is easy to write a program for calculating this polynomial and we leave it as an exercise.

In polynomial interpolation it is effective to consider cubic polynomials as these functions can be differentiated easily. Of course integration is possible. Let us consider Newton's interpolating polynomial. We do this by constructing a difference table. We will consider a polynomial of degree three.

Consider the data

x_0	x_1	x_2	x_3
$f(x_0)$	$f(x_1)$	$f(x_2)$	$f(x_3)$

We want to construct $P_3(x)$ such that

$$P_3(x) = c_0 f(x_0) + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + c_3(x - x_0)(x - x_1)(x - x_2)$$

Where

$$c_0 = f(x_0)$$

$$c_1 = \frac{[f(x_1) - f(x_0)]}{x_1 - x_0} = \Delta^{(1)}f(x_0, x_1)$$

$$c_2 = \frac{[\Delta^{(1)}f(x_1, x_2) - \Delta^{(1)}f(x_0, x_1)]}{(x_2 - x_0)} = \frac{\Delta^{(2)}f(x_0, x_1, x_2)}{(x_2 - x_0)}$$

$$c_3 = \frac{\Delta^{(2)}f(x_0, x_1, x_2, x_3)}{(x_2 - x_0)} = \frac{[\Delta^{(2)}f(x_1, x_2, x_3) - \Delta^{(2)}f(x_0, x_1, x_2)]}{(x_2 - x_0)}$$

Here, $\Delta^{(1)}f(x_0, x_1)$, $\Delta^{(2)}f(x_0, x_1, x_2)$ and $\Delta^{(3)}f(x_0, x_1, x_2, x_3)$ are called forward differences of order 1, 2 and 3 respectively. These formulae are particularly more useful if the data points are equally spaced, i.e. $\Delta x_i = x_{i+1} - x_i = h = \text{constant}$

Let us illustrate the calculation using the following data.

An example of Newton's forward interpolating polynomial

Consider the data points (x_i, y_i) that are (0, 1.0), (0.33, 1.391), (0.66, 1.935) and (1.0, 2.718). The difference table can be constructed as

x_i	$f(x_i)$	$\Delta^{(1)}f(x_i)$	$\Delta^{(2)}f(x_i)$	$\Delta^{(3)}f(x_i)$
0.0	1.000	0.39		
0.33	1.391		0.153	
		0.544		0.086
0.66	1.935		0.239	
		0.783		
1.00	2.718			

If we represent the polynomial as

$$P_3(x) = c_0 f(x_0) + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + c_3(x - x_0)(x - x_1)(x - x_2)$$

The coefficients are

$$c_0 = f(x_0), c_1 = \Delta^{(1)}f(x_0)/h = 0.391/0.33 = 1.185$$

$$c_2 = \Delta^{(2)}f(x_0)/2h^2 = 0.153/(2 \times 0.33)^2 = 0.153/0.218 = 0.702$$

$$c_3 = \Delta^{(3)}f(x_0)/6h^3 = 0.086/0.216 = 0.399$$

And the polynomial is

$$P_3(x) = 1 + 1.185(x - 0) + 0.702(x - 0)(x - 0.33) + 0.399(x - 0)(x - 0.33)(x - 0.66)$$

For an n^{th} order Newton's forward interpolating polynomial,

$$P_n(x) = \sum_{n=0}^n \frac{\Delta^n f(x_0)}{n!h^n} \prod_{i=0}^{n-1} (x - x_i)$$

Q: Determine the values of c_i by the constraint equations

$$P_n(x_i) = f(x_i)$$

e.g. $P_0(x_0) = f(x_0) \Rightarrow c_0 = f(x_0)$

$$P_1(x_1) = f(x_1) \Rightarrow f(x_1) = f(x_0) + c_1(x - x_0)$$

Or $c_1 = \frac{[f(x_1) - f(x_0)]}{x_1 - x_0} = \Delta^{(1)}f(x_0)/h$

Similarly $c_2 = \Delta^{(2)}f(x_0)/2h^2$ And $c_3 = \Delta^{(3)}f(x_0)/6h^3$

A great advantage of polynomial interpolation is that an estimate of the error can be obtained from a well-known theorem, which states that “if a function $f(x)$ defined on the interval $[a, b]$ is $(n+1)$ times differentiable, and its interpolating polynomial $P_n(x)$ interpolates it at $n+1$ distinct points $(x_0, x_1, x_2, \dots, x_n)$ on the interval $[a, b]$ then for all x^I on $[a, b]$, the error in the error in interpolation at x^I , $e_n(x^I)$ is given by

$$e_n(x^I) = f(x^I) - P_n(x^I) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^n (x^I - x_i)$$

If we know an estimate or an upper bound for the $(n+1)^{\text{st}}$ derivative $f^{(n+1)}(\xi)$, then we know the maximum possible error.

Listed below is the program for the Newton’s forward interpolation. The data files interp.dat (input file) and newtintp (output file) are listed after the fortran code.

```

c      newton's forwad interpolating polynomial(degree 3)
c      p_n(x)=y_0 + .....
c      for f(x) in the interval (x_0,x_n)
c      kth term T_k=....
c      algorithm input, initialize difference table, choose x, evaluate T_k
c      Np1 = N+1, no.of given data points at constant intervals (H)
c      DIMENSION X(10),Y(10),DELY(10,10),C(10),T(20),PN(100),XBAR(100)
c      interpolation needed at M*N +1 value of x
c      spacing for interpolated abscisa H/M =dx
c      open(unit=8, file='interp.dat')
c      open(unit=12,file='newtintp')
100  format (2I3,2F8.3)
101  format (2E10.3)
      read(8,100) NP1, M
      read(8,101) (X(I),Y(I),I=1,NP1)
      write(*,101) (X(I),Y(I),I=1,NP1)
c      read(*,*) NP1, M
c      read(*,*) (X(I),Y(I),I=1,NP1)
      H = X(2)-X(1)
c      deltax= (X(NP1)-X(1))/real(M)
      deltax=0.01
c      initialize/clear diagonal difference table
      N= NP1 - 1
      do 10 I=1,N
      do 5 K=1,N
5     DELY(I,K)=0.0
10  continue

```

```

c      generate first (order) differences deltaxi= DELY(i,1), common=
c      order of difference
      do 20 I = 1,N
20    DELY(1,I)= Y(I+1)-Y(I)
c      calculate higher order differences
      do 30 K=2,N
      NK= NP1 - K
      do 25 I=1,NK
25    DELY(K,I)=DELY ((K-1), (I+1))-DELY(K-1, I)
30    continue
      write(*,*)DELY(1,1),DELY(1,2),DELY(1,3)
      write(*,*)DELY(2,1),DELY(2,2)
      write(*,*)DELY(3,1)
      C(1)=Y(1)
      C(2)=DELY(1,1)/H
      C(3)=DELY(2,1)/(2*H*H)
      C(4)=DELY(3,1)/(6*H*H*H)
      write(*,*) C(1),C(2),C(3), C(4)
      write(*,*) M,H,deltax
c      Evaluate the polynomial at 100 points in the range between
c      X_0 and X_N
      do 50 I=1,M
      XVAR =( real(I) - 1.0) * deltax
      poly=C(1)+C(2) * (XVAR-X(1)) +C(3) * (XVAR-X(1)) * (XVAR-X(2)) +
      1C(4) * (XVAR-X(1)) * (XVAR-X(2)) * (XVAR-X(3))
      write(12,*) 'X, NEWT3POL= ', XVAR,poly
50    continue
      close(12)
      close(8)
      stop
      end

```

file interp.dat (input file) needed for the above program

```

004100
0.0000E0001.0000E000
0.3300E0001.3910E000
0.6600E0001.9350E000
0.9900E0002.7180E000

```

file newtintp (output file) resulting from running the above program

```

X, NEWT3POL= 0. 1.
X, NEWT3POL= 0.00999999978 1.0104301
X, NEWT3POL= 0.0199999996 1.02092421
X, NEWT3POL= 0.0299999993 1.0314846
X, NEWT3POL= 0.0399999991 1.04211366
X, NEWT3POL= 0.049999997 1.05281389
X, NEWT3POL= 0.0599999987 1.06358755
X, NEWT3POL= 0.0700000003 1.07443714
X, NEWT3POL= 0.0799999982 1.08536494
X, NEWT3POL= 0.0899999961 1.09637344
X, NEWT3POL= 0.099999994 1.10746503

```

X, NEWT3POL=	0.10999999	1.11864197
X, NEWT3POL=	0.11999997	1.12990689
X, NEWT3POL=	0.12999995	1.14126194
X, NEWT3POL=	0.14000001	1.15270972
X, NEWT3POL=	0.14999991	1.16425252
X, NEWT3POL=	0.15999996	1.17589271
X, NEWT3POL=	0.17000002	1.18763268
X, NEWT3POL=	0.17999992	1.19947481
X, NEWT3POL=	0.18999998	1.21142173
X, NEWT3POL=	0.19999988	1.22347546
X, NEWT3POL=	0.20999993	1.23563862
X, NEWT3POL=	0.21999999	1.2479136
X, NEWT3POL=	0.22999989	1.26030278
X, NEWT3POL=	0.23999995	1.27280843
X, NEWT3POL=	0.25	1.28543305
X, NEWT3POL=	0.25999999	1.29817915
X, NEWT3POL=	0.26999981	1.31104887
X, NEWT3POL=	0.28000001	1.3240447
X, NEWT3POL=	0.28999992	1.33716917
X, NEWT3POL=	0.29999982	1.35042453
X, NEWT3POL=	0.31000002	1.36381316
X, NEWT3POL=	0.31999993	1.37733757
X, NEWT3POL=	0.32999983	1.39100003
X, NEWT3POL=	0.34000004	1.40480304
X, NEWT3POL=	0.34999994	1.41874886
X, NEWT3POL=	0.35999985	1.43283999
X, NEWT3POL=	0.37000005	1.44707882
X, NEWT3POL=	0.37999995	1.46146762
X, NEWT3POL=	0.38999986	1.47600901
X, NEWT3POL=	0.39999976	1.49070513
X, NEWT3POL=	0.40999996	1.50555861
X, NEWT3POL=	0.41999987	1.52057171
X, NEWT3POL=	0.42999977	1.53574681
X, NEWT3POL=	0.43999998	1.55108643
X, NEWT3POL=	0.44999988	1.56659269
X, NEWT3POL=	0.45999979	1.58226836
X, NEWT3POL=	0.46999999	1.59811556
X, NEWT3POL=	0.47999989	1.6141367
X, NEWT3POL=	0.48999998	1.63033426
X, NEWT3POL=	0.5	1.64671063
X, NEWT3POL=	0.50999999	1.66326821
X, NEWT3POL=	0.51999981	1.68000925
X, NEWT3POL=	0.52999971	1.69693625
X, NEWT3POL=	0.53999962	1.71405172
X, NEWT3POL=	0.55000012	.73135793
X, NEWT3POL=	0.56000002	1.74885726
X, NEWT3POL=	0.56999993	1.76655209
X, NEWT3POL=	0.57999983	1.78444493
X, NEWT3POL=	0.58999974	1.80253804
X, NEWT3POL=	0.59999964	1.8208338
X, NEWT3POL=	0.61000014	1.83933485
X, NEWT3POL=	0.62000005	1.85804331
X, NEWT3POL=	0.62999995	1.87696159
X, NEWT3POL=	0.63999986	1.89609218
X, NEWT3POL=	0.64999976	1.91543746
X, NEWT3POL=	0.65999967	1.93499982

X, NEWT3POL=	0.669999957	1.95478165
X, NEWT3POL=	0.680000007	1.97478545
X, NEWT3POL=	0.689999998	1.99501336
X, NEWT3POL=	0.699999988	2.01546788
X, NEWT3POL=	0.709999979	2.03615165
X, NEWT3POL=	0.719999969	2.05706668
X, NEWT3POL=	0.729999959	2.0782156
X, NEWT3POL=	0.74000001	2.09960079
X, NEWT3POL=	0.75	2.1212244
X, NEWT3POL=	0.75999999	2.14308929
X, NEWT3POL=	0.769999981	2.16519737
X, NEWT3POL=	0.779999971	2.18755126
X, NEWT3POL=	0.789999962	2.21015334
X, NEWT3POL=	0.799999952	2.233006
X, NEWT3POL=	0.810000002	2.25611186
X, NEWT3POL=	0.819999993	2.27947283
X, NEWT3POL=	0.829999983	2.30309153
X, NEWT3POL=	0.839999974	2.32697058
X, NEWT3POL=	0.849999964	2.35111189
X, NEWT3POL=	0.859999955	2.37551832
X, NEWT3POL=	0.870000005	2.40019226
X, NEWT3POL=	0.879999995	2.42513561
X, NEWT3POL=	0.889999986	2.45035124
X, NEWT3POL=	0.899999976	2.47584128
X, NEWT3POL=	0.909999967	2.50160813
X, NEWT3POL=	0.919999957	2.52765441
X, NEWT3POL=	0.930000007	2.5539825
X, NEWT3POL=	0.939999998	2.58059454
X, NEWT3POL=	0.949999988	2.60749316
X, NEWT3POL=	0.959999979	2.63468051
X, NEWT3POL=	0.969999969	2.66215897
X, NEWT3POL=	0.979999959	2.68993139
X, NEWT3POL=	0.98999995	2.7179997

4.6 CURVE FITTING

Fitting a ‘good’ curve through a given set of data points is a very important numerical method. It is often better to fit a smooth function passing through most of the data points as the fit may be better than an interpolating polynomial. A smooth fit may eliminate some of the random errors of the experimental data. Let the fitting function be called $F(x)$. The fitting function could be a polynomial such as

$$F(x) = a_0 + a_1x + a_2x^2 + \dots + a_mx^m$$

In place of x, x^2, x^3 , we could have other functions $f_1(x), f_2(x), f_3(x)$, etc.

Let us tabulate the values as follows

x	y	$F(x)$	$r_i = F(x_i) - y_i$	
	x_0	y_0	$F(x_0)$	$r_0 = F(x_0) - y_0$
x_1	y_1	$F(x_1)$	$r_1 = F(x_1) - y_1$	
	.			
	.			
	.			
x_n	y_n	$F(x_n)$	$r_n = F(x_n) - y_n$	

The last column is called as the residual, which is the difference between the fitting function and the dependent/measured variable y . For a good fit, the sum of the square of the residuals, S must be as small as possible, i.e.

$$S = \sum_{i=0}^n r_i^2 = \sum_{i=0}^n (y_i - F(x_i))^2 \rightarrow \text{minimum}$$

This can be achieved by setting

$$\frac{\partial S}{\partial a_i} = 0, \forall i = 0, \dots, m$$

For the polynomial function,

$$\frac{\partial S}{\partial a_k} = \frac{\partial}{\partial a_k} \sum_{i=1}^n [a_0 + a_1 x + a_2 x^2 + \dots + a_k x_i^k - y_i]^2 = 0$$

$$\text{Or } 2 \sum_{i=1}^n [a_0 + a_1 x + a_2 x^2 + \dots + a_m x_i^m - y_i] x_i^k = 0 \quad \forall k = 0, \dots, m$$

Multiply through by x_i^k and eliminate the factor of 2 to get

$$a_0 \sum x_i^k + a_1 \sum x_i x_i^k + a_2 \sum x_i^2 x_i^k + \dots + a_m \sum x_i^m x_i^k = \sum y_i x_i^k \quad \text{for } k = 0, \dots, m$$

The equations for the coefficients a_0, a_1, \dots, a_m may be written in a matrix form as

$$\begin{bmatrix} \sum(x_i^0) & \sum x_i^0 x_i & \sum x_i^0 x_i^2 & \dots & \dots & \sum x_i^0 x_i^m \\ \sum(x_i) & \sum x_i x_i & \sum x_i x_i^2 & \dots & \dots & \sum x_i x_i^m \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum(x_i^m) & \sum x_i^m x_i & \sum x_i^m x_i^2 & \dots & \dots & \sum x_i^m x_i^m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^m y_i \end{bmatrix}$$

To solve this equation, we need the matrix method which is taken up in the next chapter. Instead of x, x^2, \dots we could have had any other functions. In this scheme, we are determining the linear parameters a_i . If we are fitting a straight line through the points, this is called linear regression and the equations are quite simple.

$$\begin{bmatrix} \sum(x_i^0) & \sum x_i^0 x_i \\ \sum(x_i) & \sum x_i x_i \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

Solving for a_0 and a_1 we get

Example: obtain the third degree polynomial for the following data

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	0.0	0.1002	0.2013	0.3045	0.4108	0.5211	0.6367	0.7586	0.8881	1.0265	1.1752

$$P_3(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3$$

The quantities needed for constructing the normal equations:

$$n + 1 = 11, \quad \sum y_i = 6.023$$

$$\sum(x_i) = 5.5, \quad \sum x_i y_i = 4.28907$$

$$\sum(x_i^2) = 3.85, \quad \sum x_i^2 y_i = 3.408407$$

$$\sum(x_i^3) = 3.025, \quad \sum x_i^3 y_i = 2.8773135$$

$$\sum(x_i^4) = 2.5333$$

$$\sum(x_i^5) = 2.20825$$

$$\sum(x_i^6) = 1.987405$$

$$\begin{bmatrix} 11.0 & 5.5 & 3.85 & 3.025 \\ 5.5 & 3.85 & 3.025 & 2.5333 \\ 3.85 & 3.025 & 2.5333 & 2.20825 \\ 3.025 & 2.5333 & 2.20825 & 1.987405 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 6.023 \\ 4.28907 \\ 3.408407 \\ 2.8773135 \end{bmatrix}$$

The solution of the normal equations:

$$a_1 = -0.000129, a_2 = 1.004383, a_3 = -0.019651, a_4 = 0.190405$$

The third degree least squares polynomial is

$$P_3(x) = -0.000129 + 1.004383x - 0.019651x^2 + 0.190405x^3$$

The above data is for the function $f(x) = \sinh(x) = x + \frac{x^3}{3!} + \dots$

For higher order polynomials and especially for unevenly spaced points the determinant tends to be $\rightarrow 0$ and create problems.

4.7 Summary

In this chapter, we considered the numerical methods involving interpolation and curve fitting. Both are very good ways for representing data in a compact form which is useful for other calculations such as obtaining the integrals or derivatives of these functions. In interpolation, the curve has to pass through the

data points while in the fitted function; it need not pass through the points, but we try to get the 'smoothest' curve passing "near" the points so that the error or the residual between the fitted values and the data is a minimum. We have considered fitting functions with linear parameters. If we need to find fitting functions such as $e^{-a_1x} + e^{-a_2x}$ or $\sin a_1x + \cos a_2x$, note that in these functions, the parameters a_1, a_2 are nonlinear and the problem is a little more difficult and the results are not unique as well. In polynomial interpolation, it is best to use low order polynomials of order 2 or 3. If the data extends to say 12 points, it is better to fit four cubic polynomials between points.

$(x_0 \text{ to } x_3), x_3 \text{ to } x_6 \text{ and } x_6 \text{ to } x_9 \text{ and } x_9 \text{ to } x_{12}$. Fitting a very high order polynomial leads to rapidly oscillating functions which are generally unrealistic and therefore, higher order (higher than 3) polynomials are not generally recommended.