Numerical Analysis for Chemical Engineers

Taechul Lee (tclee@prosys.korea.ac.kr)

April 18, 2002

Contents

1 Modeling, Computers, and Error Analysis **2**

1.1 Mathematical Modeling and Engineering Problem-Solving 2

1.1.1 A Simple Mathematical Model 2

1.2 Computers and Software 3

1.2.1 The Software Development Process 3

1.2.2 Algorithm Design 3

1.2.3 Program Composition 3

1.2.4 Quality Control 4

1.3 Approximations and Round-Off Errors 4

1.3.1 Significant Figures 4

1.3.2 Accuracy and Precision 4

1.3.3 Error Definitions 4

1.4 Truncation Errors and the Taylor Series 5

1.4.1 The Taylor Series 5

1.4.2 Using the Taylor Series to Estimate Truncation Errors 7

1.4.3 Numerical Differentiation 7

2 Roots of Equations **9**

2.1 Backeting Methods 9

2.1.1 Graphical Methods 9

2.1.2 The Bisection Method . . . 9
2.1.3 The False-Position Method . . . 9
2.2 Open Methods . . . 10
2.2.1 Simple Fixed-point Iteration . . . 10
2.2.2 The Newton-Raphson Method . . . 10
2.2.3 The Secant Method . . . 11
2.2.4 Multiple Roots . . . 12
2.2.5 Systems of Nonlinear Equations . . . 12
2.3 Roots of Polynomials . . . 13
2.3.1 Polynomials in Engineering and Science . . . 13
2.3.2 Computing with Polynomials . . . 14
2.3.3 Conventional Methods . . . 15
2.3.4 Root Location with Libraries and Packages . . . 15
2.4 Engineering Applications: Roots of Equations . . . 15

3 Linear Algebraic and Equations 16
3.1 Gauss Elimination . . . 16
3.1.1 Solving Small Numbers of Equations . . . 16
3.1.2 Naive Gauss Elimination . . . 17
3.1.3 Pitfalls of Elimination Methods . . . 17
3.1.4 Techniques for Improving Solutions . . . 18
3.1.5 Complex Systems . . . 18
3.1.6 Nonlinear Systems of Equations . . . 18
3.1.7 Gauss-Jordan . . . 19
3.2 LU Decomposition and Matrix Inversion . . . 19
3.2.1 LU Decomposition . . . 19
3.2.2 The Matrix Inverse . . . 21
3.2.3 Error Analysis and System Condition . . . 21

3.3 Special Matrices and Gauss-Seidel 22
3.3.1 Special Matrices 22
3.3.2 Gauss-Seidel 22
3.3.3 Linear Algebraic Equation with Libraries and Packages 22
3.4 Engineering Applications: Linear Algebraic Equations 23

4 Optimization **24**
4.1 One-dimensional Unconstrained Optimization 25
4.1.1 Golden-Section Search 25
4.1.2 Quadratic Interpolation 26
4.1.3 Newton's Method 26
4.2 Multidimensional Unconstrained Optimization 27
4.2.1 Direct Methods 27
4.2.2 Gradient Methods 28
4.3 Contrained Optimization 29
4.3.1 Linear Programming 29
4.3.2 Optimization with Packages 30
4.4 Engineering Applications: Optimization 30

5 Curve Fitting **31**
5.1 Least-Squares Regression 32
5.1.1 Linear Regression 32
5.1.2 General Linear Least-Squares 34
5.1.3 Nonlinear Regression 35
5.2 Interpolation 35
5.2.1 Newton's Divided-Difference Interpolating Polynomials 35
5.2.2 Lagrange Interpolating Polynomial 36
5.2.3 Spline Interpolation 36

5.3 Fourier Approximation 37
5.3.1 Curve Fitting with Sinusoidal Functions 41
5.3.2 Fourier Integral and Transform 41
5.3.3 Discrete Fourier Transform (DFT) 44
5.3.4 Fast Fourier Transform (FFT) 44
5.3.5 The Power Spectrum 45
5.3.6 Curve Fitting with Libraries and Packagies 45
5.4 Engineering Applications: Curve Fitting 45

6 Numerical Differentiation and Integration **46**
6.1 Newton-Cotes Integration of Equations 46
6.1.1 The Trapezoidal rule 47
6.1.2 Simpson's rule 48
6.2 Intergrations of Equations 48
6.2.1 Romberg integration 48
6.2.2 Gauss Quadrature 49
6.2.3 Improper integrals 51
6.3 Numerical Differentiation 51
6.3.1 High-accuracy differentiation formulas 51
6.3.2 Richardson extrapolation 52
6.3.3 Derivatives of unequally spaced data 52
6.3.4 Numerical integration/differentiation formulas with libraties and packages . 52
6.4 Engineering Applications: Numerical Integration and Differentiation 52

7 Ordinary Differential Equations **53**
7.1 Runge-Kutta Methods 54
7.1.1 Euler's Method 54
7.1.2 Improvement of Euler's Method 55

7.1.3 Runge-Kutta Method 55
7.1.4 Systems of Equations 58
7.1.5 Adaptive Runge-Kutta Method 59
7.2 Stiffness and Multistep Methods 59
7.2.1 Stiffness 59
7.2.2 Multistep Methods 60
7.3 Boundary-Value and Engenvalue Problems 63
7.3.1 General Methods of Boundary-Value Problems 63
7.3.2 ODEs and Eigenvalues with Libraries and Packages 64
7.4 Engineering Applications: Ordinary Differential Equations 64

8 Partial Differential Equations **65**
8.1 Finite Difference: Elliptic Equations 66
8.1.1 The Laplace Equations 66
8.1.2 Solution Techniques 66
8.1.3 Boundary Conditions 67
8.1.4 The Control Volume Approach 67
8.2 Finite Difference: Parabolic Equations 67
8.2.1 The Heat Conduction Equation 67
8.2.2 Explicit Methods 68
8.2.3 A Simple Implicit Method 70
8.2.4 The Crank-Nicholson Method 71
8.3 Finite Element Method 71
8.3.1 Calculus of variation 71
8.3.2 Example: The shortest distance between two points 73
8.3.3 The Rayleigh-Ritz Method 75
8.3.4 The Collocation and Galerkin Method 76
8.3.5 Finite elements for ordinary-differential equations 77

8.4 Engineering Applications: Partial Differential Equations 77

A Using Matlab **78**

A.1 설치 78

A.2 Matlab 기초 79

A.2.1 배열 83

A.2.2 Customization 86

A.2.3 Summary 89

A.3 제어문 89

A.3.1 if, else, and elseif 89

A.3.2 switch 91

A.3.3 while 92

A.3.4 for 93

A.3.5 break 94

A.3.6 Summary 94

A.4 함수만들기 95

A.5 Matlab에서 그림 그리기 98

A.5.1 plot 명령어 98

A.5.2 고급 plot 명령어 106

A.5.3 그림을 그리는 다른 명령어들 115

A.6 예제 125

A.6.1 Linear Equation 125

B Using Fortran **130**

B.1 설치 및 사용법 130

B.1.1 MS Window에서 작동하는 포트란 130

B.1.2 Unix 머신에서 작동하는 포트란 132

B.1.3 Summary 133

B.2 데이터와 입출력 134
B.2.1 기본적 구성 134
B.2.2 기본적 데이터 타입 134
B.2.3 입력과 출력에 관해 139
B.2.4 Redirection 143
B.2.5 Dimension 148
B.2.6 데이터 초기화 149
B.2.7 Summary 151
B.3 제어문 151
B.3.1 STOP문 151
B.3.2 GOTO문 152
B.3.3 PAUSE문 152
B.3.4 CONTINUE문 152
B.3.5 CALL문 153
B.3.6 RETURN문 153
B.3.7 IF문 153
B.3.8 DO문 157
B.3.9 Summary 162
B.4 부프로그램 163
B.4.1 FUNCTION 163
B.4.2 SUBROUTINE 167
B.4.3 부프로그램 컴파일 175
B.4.4 라이브러리 만들기 179
B.4.5 EXTERNAL 문 사용하기 182
B.4.6 IMSL 사용하기 186

Abstract

수치해석이라고 하는 과목은 간단히 얘기를 하면 수학식으로 표현된것을 실제 우리가 볼 수 있는 형태로 구현하는 방법에 대한 과목이다. 또한 이 수치해석이 필요한 이유는 대부분 실제 우리가 볼 수 있는 현상은 수학식으로 표현할 수 있고 이렇게 표현된 식은 식 자체만으로는 쉽게 이해하기가 힘들다. 그러므로 수치해석이라는 과정을 거쳐서 실제화 한다고 볼 수 있다. 그렇지만 수치해석 방법들은 대부분은 간단하고 단순한 많은 계산 과정으로 구성이 되어 있다. 그러므로 이러한 간단하고 단순한 과정을 사람대신 해결할 수 있는 방법은 컴퓨터를 이용하는 방법이다. 컴퓨터가 발전하고 계산 속도가 향상됨에 따라서 수치해석이라는 부분에 대한 관심도 증가되고 있고 기존에 시도해보지 못한 여러가지 방법들이 시도되고 있다.

이 과목에서는 자연현상을 수학적 모델로서 구성을 하고 이 모델을 수치해석 기법을 이용해서 컴퓨터로서 해결하는 방법에 대해 다룬다. 모델을 만는 것은 여러분들이 이미 오래전부터 배워왔던 부분일 것이며 수치해석이라는 것도 또한 여러분들이 종이와 손을 이용해서 해결했던 방법을 대부분 컴퓨터로 적용을 하는 것이다. 사용하는 언어로는 오랫동한 수치해석에서 많이 쓰였던 포트란과 최근 10년동안 많이 쓰이고 있는 Matlab을 이용할 것이다. 두 언어에 대한 것은 APPENDIX 부분을 참고하기 바란다.

Chapter 1

Modeling, Computers, and Error Analysis

1.1 Mathematical Modeling and Engineering Problem-Solving

1.1.1 A Simple Mathematical Model

수학적 모델이라는 것은 수학적 용어로서 물리적 자연현상의 중요한 부분을 식으로서 구성하는 것이다. 이러한 수학모델을에서 결과를 얻는 방법은 크게 두가지가 있다.

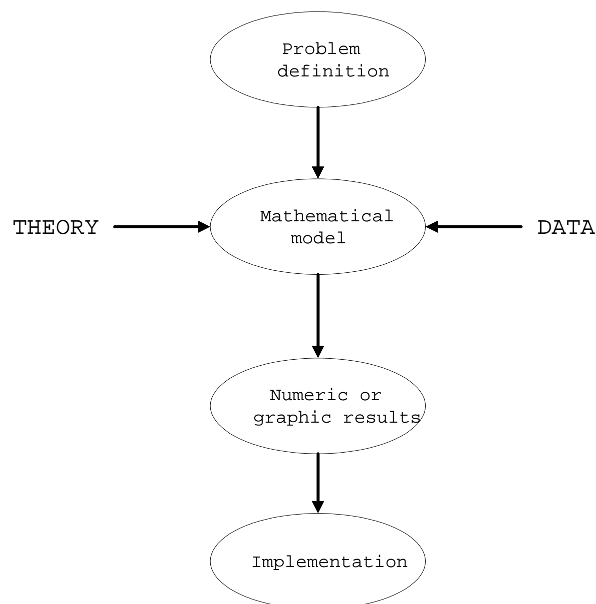

Figure 1.1: The engineering problem-solving process.

- Analytical or exact solution
- Numerical solution

위 두가지 가운데 해석적 해를 얻는 방법은 다룰 수 있는 문제가 한정되어 있고 또한 대부분의 실제 문제는 비선형적이고 복잡한 과정이 포함되어 있으므로 적절하지 않다. 그러므로 수치적 해를 구할 수 밖에 없다.

1.2 Computers and Software

1.2.1 The Software Development Process

- Programming Style
- Modular Design : Devide into small subprograms
- Top-down Design : Sysmtematic development process
- Structured Programming : How the actual program code is developed

1.2.2 Algorithm Design

- Flowschart : a visual or graphical representation of an algorithm
- Pseudocode : bridges the gap between flowcharts and computer code

1.2.3 Program Composition

- High-level and Macro Languages : C, Fortran, Basic
- Structured Programming
  - consist of the three funcdamental control structures of sequence, selection, and repetition
  - only one entrance and one exit
  - Unconditional transfers should be avoided
  - identified with comments and visual devices such as indentation, blank lines, and blank spaces

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

1.2.4 Quality Control

- Errors or “Bugs”
  - Syntax errors
  - Link or build errors
  - Run-time errors
  - Logic errors
- Debugging
- Testing

1.3 Approximations and Round-Off Errors

1.3.1 Significant Figures

Significant figure : The reliability of a numerical value

1.3.2 Accuracy and Precision

- Accuracy : How closely a computed or measured value agrees with the true value
- Precision : How closely individual computed or measured values agree with each other

1.3.3 Error Definitions

- Truncation error : approximations are used to represent exact mathematical prodedures
- Round-off error : numbers having limited significant figures are used to represent exact numbers

See Figure 3.10, 3.11 and 3.12 in the textbook.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

1.4 Truncation Errors and the Taylor Series

1.4.1 The Taylor Series

If a function $f(x)$ can be represent by a power series on the interval $(-a, a)$, then the function has derivative of all orders on that interval and the power series is

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots \tag{1.1}$$

and this power-series expansion of $f(x)$ about the origin is called a Maclaurin series.

If the expansion is about the point $x = a$, we have the Taylor series

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(0)}{3!}(x-a)^3 + \dots \tag{1.2}$$

Taylor series specifies the value of function at one point, x, in terms of the value of the function and its derivatives at a reference point, a. It is occasionally useful to express a Taylor series in a notation that show how the function behaves at a distance h from a fixed point a. If we call $x = a + h$ in the preceding series, so that $x - a = h$, we get

$$f(a+h) = f(a) + f'(a)h + \frac{f''(a)}{2!}h^2 + \frac{f'''(0)}{3!}h^3 + \dots \tag{1.3}$$

Or with the substitution $a + h \to x_{i+1}$ and $a \to x_i$ we have an alternate form

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \dots + \frac{f^{(n)}(x_i)}{n!}(x_{i+1} - x_i)^n + R_n \tag{1.4}$$

R_n term is a reminder term to account for all terms from $n + 1$ to infinity:

$$R_n = \frac{f^{(n+1)}(\zeta)}{(n+1)!}(x_{i+1} - x_i)^{n+1} \tag{1.5}$$

- Mean-value theorem:
  If a function $f(x)$ and its first derivative are continuous over an interval from x_i and x_{i+1}, then there exists at least one point on the function that has a slope, designated by $f'(\xi)$, that is parallel to the line joining $f(x_i)$ and $f(x_{i+1})$.

See Figure 4.3 in the textbook.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

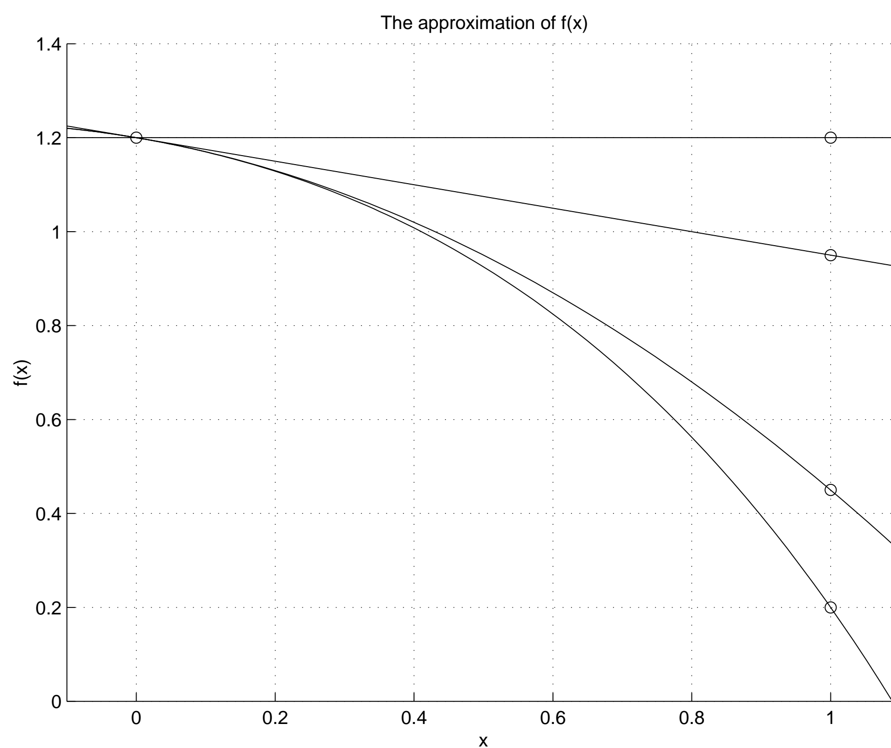

Figure 1.2: The approximation of $f(x)$ with various order of Taylor series.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

1.4.2 Using the Taylor Series to Estimate Truncation Errors

Taylor series expansion of $v(t)$:

$$v(t_{i+1}) = v(t_i) + v'(t_i)(t_{i+1} - t_i) + \frac{v''(t_i)}{2!}(t_{i+1} - t_i)^2 + \cdots + R_n \tag{1.6}$$

Truncate the series after the first derivative term:

$$v(t_{i+1}) = v(t_i) + v'(t_i)(t_{i+1} - t_i) + R_1 \tag{1.7}$$

And

$$v'(t_i) = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} - \frac{R_1}{t_{i+1} - t_i} \tag{1.8}$$

Truncation error is

$$\frac{R_1}{t_{i+1} - t_i} = \frac{v''(\xi)}{2!}(t_{i+1} - t_i) \tag{1.9}$$

or

$$\frac{R_1}{t_{i+1} - t_i} = O(t_{i+1} - t_i) \tag{1.10}$$

The error of our derivative approximation should be proportional to the step size. Consequently, if we halve the step size, we would expect to halve the error of the derivative.

1.4.3 Numerical Differentiation

- Forward Difference Approximation

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} + O(t_{i+1} - t_i) \tag{1.11}$$

  or

$$f'(x_i) = \frac{\Delta f_i}{h} + O(h) \tag{1.12}$$

  where Δf_i is the first forward difference.

- Backward Difference Approximation

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(t_i - t_{i-1}) \tag{1.13}$$

  or

$$f'(x_i) = \frac{\nabla f_i}{h} + O(h) \tag{1.14}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- Central Difference Approximation

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2) \tag{1.15}$$

Notice that the truncation error is of the order of h^2 in contrast to the forward and backward approximations that were of the order of h. Consequently, the Taylor series analysis yields the practical information that the centered difference is a more accurate representation of the derivative.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 2

Roots of Equations

2.1 Backeting Methods

2.1.1 Graphical Methods

A simple method for obtaining an estimate of the root of the equation $f(x) = 0$ is to make a plot of the function and observe where it crosses the x axis.

2.1.2 The Bisection Method

In general, if $f(x)$ is real and continuous in the interval from x_l to x_u and $f(x_l)$ and $f(x_u)$ have opposite signs, that is

$$f(x_l)f(x_u) < 0 \tag{2.1}$$

then there is at least one real root between x_l and x_u.

2.1.3 The False-Position Method

A shortcoming of the bisection method

- equally dividing the interval
- no account for for the magnitudes of $f(x_l)$ and $f(x_u)$

An alternative method is to join $f(x_l)$ and $f(x_u)$ by a straight line and the intersection of this line with the x axis represents an improved estimate of the root. This mothod is called as method of false position, regula falsi, or linear interpolation method.

The false-position formula is

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)} \tag{2.2}$$

See Figure 5.14 in textbook

2.2 Open Methods

- bracketing method : the root is located within an interval prescribed by a lower and an upper bound.
- open method : require only a single starting value of x or two starting point that do not necessarily bracket the root.

2.2.1 Simple Fixed-point Iteration

Open methods employ a formula to predict the root. Such a formula can be develped for simple fixed-point iteration by rearranging the function $f(x) = 0$ so that s is on the left-hand side of the equation:

$$x = g(x) \tag{2.3}$$

2.2.2 The Newton-Raphson Method

If the initial guess at the root is x_i, a tangent can be extended from the point $[x_i, x(x_i)]$. The point where this tangent crosses the x axis usaually represents an improved estimate of the root.

The Newton-Raphson formula is

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2.4}$$

Pitfalls of the Newton-Raphson Method are shown in Figure 6.6

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

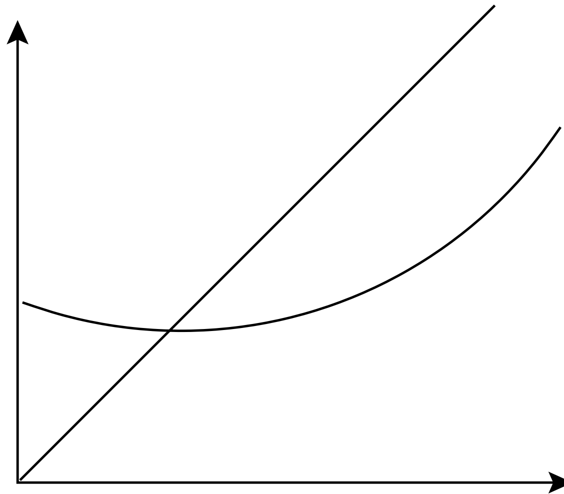

Figure 2.1: Graphical depiction of simple fixed-point method.

2.2.3 The Secant Method

A potential problem in implementing the Newton-Raphson method is the evaluation of the derivative. In Secant method the derivative is approximated by a backward finite divided difference

$$f'(x_i) \simeq \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i} \tag{2.5}$$

The Secant formula is

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \tag{2.6}$$

The difference between the secant method and the false-position method is how one of the initial values is replaced by the new estimate.

Rather than using two arbitrary values to estimate the derivative, an alternative approach involves a fractional perturbation of the independent variable to estimate $f'(x)$,

$$f'(x_i) \simeq \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i} \tag{2.7}$$

where δ is a small perturbation fraction. This approximation gives the following iterative equation:

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)} \tag{2.8}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

2.2.4 Multiple Roots

Some difficulities in multiple roots problem

- no change in sign at even multiple roots
- $f(x)$ and $f'(x)$ go to zero at the root
- the Newton-Raphson method and secant method show linear, rather than quadratic, convergence for multiple roots.

Another alternative is to define a new function $u(x)$,

$$u(x) = \frac{f(x)}{f'(x)} \tag{2.9}$$

An alternative form of the Newton-Raphson method:

$$x_{i+1} = x_i - \frac{u(x)}{u'(x)} \tag{2.10}$$

where $u'(x)$ is

$$u'(x) = \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} \tag{2.11}$$

And finally

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)} \tag{2.12}$$

2.2.5 Systems of Nonlinear Equations

The Newton-Raphson method can be used to solve a set of nonlinear equations. The Newton-Raphson method employ the derivative of a function to estimate its intercept with the axis of the independent variable. This estimate was based on a first-order Taylor series expansion. For example, we consider two variable case,

$$u(x, y) = 0 \tag{2.13}$$

$$v(x, y) = 0 \tag{2.14}$$

A first-order Taylor series expasion can be written as

$$u_{i+1} = u_i + (x_{i+1} - x_i)\frac{\partial u_i}{\partial x} + (y_{i+1} - y_i)\frac{\partial u_i}{\partial y} \tag{2.15}$$

$$v_{i+1} = v_i + (x_{i+1} - x_i)\frac{\partial v_i}{\partial x} + (y_{i+1} - y_i)\frac{\partial v_i}{\partial y} \tag{2.16}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

The above equation can be rearranged to give

$$\frac{\partial u_i}{\partial x}x_{i+1} + \frac{\partial u_i}{\partial y}y_{i+1} = -u_i + x_i\frac{\partial u_i}{\partial x} + y_i\frac{\partial u_i}{\partial y} \tag{2.17}$$

$$\frac{\partial v_i}{\partial x}x_{i+1} + \frac{\partial v_i}{\partial y}y_{i+1} = -v_i + x_i\frac{\partial v_i}{\partial x} + y_i\frac{\partial v_i}{\partial y} \tag{2.18}$$

Finally

$$x_{i+1} = x_i - \frac{u_i\frac{\partial v_i}{\partial y} - v_i\frac{\partial u_i}{\partial y}}{\frac{\partial u_i}{\partial x}\frac{\partial v_i}{\partial y} - \frac{\partial u_i}{\partial y}\frac{\partial v_i}{\partial x}} \tag{2.19}$$

$$y_{i+1} = y_i - \frac{v_i\frac{\partial u_i}{\partial x} - u_i\frac{\partial v_i}{\partial x}}{\frac{\partial u_i}{\partial x}\frac{\partial v_i}{\partial y} - \frac{\partial u_i}{\partial y}\frac{\partial v_i}{\partial x}} \tag{2.20}$$

2.3 Roots of Polynomials

The roots of polynomials have the following properties

- For an nth-order equation, there are n real and complex roots.
- If n is odd, there is at least one real root.
- If complex roots exist, they exist in conjugate pairs.

2.3.1 Polynomials in Engineering and Science

Polynomial are used extensively in curve-fitting. However, another most important application is in characterizing dynamic system and, in particular, linear systems.

For example, we consider the following simple second-order ordinary differential equation:

$$a_2\frac{d^2y}{dt} + a_1\frac{dy}{dt} + a_0y = F(t) \tag{2.21}$$

where $F(t)$ is the forcing function. And the above ODE can be expressed as a system of 2 first-order ODEs by defining a new variable z,

$$z = \frac{dy}{dt} \tag{2.22}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

This reduces the problem to solving

$$\frac{dz}{dt} = \frac{F(t) - a_1 z - a_0 y}{a_2} \tag{2.23}$$

$$\frac{dy}{dt} = z \tag{2.24}$$

In a simliar fashion, nth-order linear ODE can always be expressed as a system of n first-order ODEs.

The general solution of ODE equation deals with the case when the forcing function is set to zero.

$$a_2 \frac{d^2 y}{dt} + a_1 \frac{dy}{dt} + a_0 y = 0 \tag{2.25}$$

This equation gives something very fundamental about the system being simulated-that is, how the system reponds in the absence of external stimuli. The general solution to all unforced linear system is of the form $y = e^{rt}$.

$$a_2 r^2 e^{rt} + a_1 r e^{rt} + a_0 e^{rt} = 0 \tag{2.26}$$

or cancelling the exponential terms,

$$a_2 r^2 + a_1 r + a_0 = 0 \tag{2.27}$$

This polynomial is called as characteristic equation and these r's are referred to as eigenvalues.

$$\begin{matrix} r_1 \\ r_2 \end{matrix} = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2 a_0}}{a_0} \tag{2.28}$$

- overdamped case : all real roots
- critically damped case : only one root
- underdamped case : all complex roots

2.3.2 Computing with Polynomials

For nth-order polynomial calculation, general approach requires $n(n+1)/2$ multiplications and n additions. However, if we use a nested format, n multiplications and n additions are required.

```
DO j=n, 0
   p = p * x + a(j)
END DO
```

If you want to find all roots of a polynomial, you have to remove the found root before another processing. This removal process is referred to as polynomial deflation.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

2.3.3 Conventional Methods

- Müller's method : projects a parabola through three points.
- Bairstow's method:
  1. guess a value for the root $x = t$
  2. divide the polynomial by the factor $x - t$
  3. determine whether there is a reminder. If not, the guess is was perfect and the root is equal to. If there is a reminder, the guess can be systematically adjusted and the procedure repeated until the reminder disappears.

2.3.4 Root Location with Libraries and Packages

- Matlab:
  - roots
  - poly
  - polyval
  - residue
  - conv
  - deconv
- IMSL:
  - ZREAL

2.4 Engineering Applications: Roots of Equations

See the textbook

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 3

Linear Algebraic and Equations

Matrix notation:

- sysmmetric matrix
- diagonal matrix
- principal or main diagonal of the matrix
- identity matrix
- upper triangular matrix
- lower triangular matrix
- banned matrix
- tridiagonal matrix
- transpose
- trace

3.1 Gauss Elimination

3.1.1 Solving Small Numbers of Equations

Problems when solving sets of linear equations

- Singular
  - no solution
  - infinite solution
- Ill-conditioned : system that are very close to being singular

Cramer's Rule : each unknown in a system of linear algebraic equations may be expressed as a fraction of two determinants with denominator D and with the numerator obtained from D by replacing the column of coefficients of the unknown in question by the constant $b_1, b_2, \ldots, b_n$. For more than three equations, Cramer's rule becomes impractical because, as the number of equations increases, the determinants are time-consuming to evaluate.

3.1.2 Naive Gauss Elimination

The elimination of unknowns consists of two steps

1. The equations are manipulated to eliminate one of the unknowns from the equations. The result of this elimination step is that we have on equation with one unknown.
2. This equation can be solved directly and the result back-substituted into one of the original equations to solve for the remaining unknown.

But the above method can't avoid division by zero in computer program. Need more elaborated algorithms !

3.1.3 Pitfalls of Elimination Methods

- Division by zero : partially avoided by the technique of pivoting
- Round-off errors : An error in early steps wiil tend to propagate-that is, it will cause errors in subsequent steps.
- Ill-conditioned systems : small changes in coefficients result in large changes in the solution. An ill-conditioned system is one with a determinant close to zero. This means that there is no solutions or an infinite number of solutions. However, it is difficult to specify how close to zero the determinant must be to indicate ill-conditioning. This is complicated by the fact that the determinant can be changed by multiplying one or more of the equations by a scale factor without changing the solution. Consequently, the determinant is a relative value that is influenced by the magnitude of the coefficients. One way to partially prevent a scaling effect is to scale the equations so that the maximum element in any row is equal to 1.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- Singular systems : lose one degree of freedom and we would be dealing with the impossible case of $n-1$ equations with n unknowns. Check the determinant whether it is zero or not !

3.1.4 Techniques for Improving Solutions

- Use more significant figures in the computation
- Use pivoting
  - partial pivoting : make the largest element in the column to be the pivot element
  - avoiding division by zero
  - minimizes round-off error
  - gives a partial remedy for ill-conditioning
- Use scaling : minimize round-off error for those cases where some of the equations in a system have much larger coefficients than others.

However, sometime scaling introduces a round-off error. Thus, it is used as a criterion for pivoting and the original coefficient values are retrained for the actual elimination and substitution computation.

3.1.5 Complex Systems

Convert a complex system into two real system and employ the algorithm for the real system.

3.1.6 Nonlinear Systems of Equations

Use a multidimensional verstion of Newton-Raphson method for nonlinear system. However, there are two major shortcoming.

- Difficult to calculate partial derivatives
- Need excellent initial guesses

Need optimization techniques !

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

3.1.7 Gauss-Jordan

Difference between Gauss and Gauss-Jordan

- An unknown is elminated from all other euqation rather than just the subsequent ones.
- All rows are normalized by dividing them by their pivot elements.
  - the elimination step results in an identity matrix rather than a triangular matrix
  - No need to employ back substitution to obtain the solution.

3.2 LU Decomposition and Matrix Inversion

LU decomposition provides

- well-suited for those situations where many right-hande side vector $\{B\}$ must be evaluated for a single value of $[A]$.
- an efficient means to compute the matrix inverse.

3.2.1 LU Decomposition

Gauss elimination is designed to solve system of linear algebraic equations

$$[A]\{X\} = \{B\} \tag{3.1}$$

Gauss elimination involves two steps: forward elimination and back substitution. Of these, the forward elimination step require more computation times. LU decomposition methods separate the time-consuming elimination of the matrix $[A]$ from the manipulations of the right-hand side $\{B\}$. Thus, once $[A]$ has ben decomposed, multiple right-hand side vectors can be evaluated in an efficient manner.

Equation (3.1) can be rearranged to give

$$[A]\{X\} - \{B\} = 0 \tag{3.2}$$

The above equation can be reduced into upper triangular form with Gauss elimination.

$$[U]\{X\} - \{D\} = 0 \tag{3.3}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

And assume a lower triangular matrix $[L]$ which satisfies the following equation

$$[L]\{[U]\{X\} - \{D\}\} = [A]\{X\} - \{B\} \tag{3.4}$$

If this equation holds,

$$[L][U] = [A] \tag{3.5}$$
$$[L][D] = \{B\} \tag{3.6}$$

A two-step strategy for obtaining solutions with LU decomposition

1. Decompose $[A]$into $[L]$ and $[U]$
2. Find a intermediate vector $\{D\}$ with equation (3.6) and solve equation (3.3) for $\{X\}$

LU decomposition can be performed in Gauss elimination. $[U]$ is a direct product of the forward elimination. For example, consider the following 3×3 system

$$\Big[A\Big] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{3.7}$$

The forward elimination step reduce the original matrix $[A]$ to the form

$$\Big[U\Big] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \tag{3.8}$$

which is in the desired upper triangular format. The matrix $[L]$ is also produced during the step.

$$\Big[L\Big] = \begin{bmatrix} 1 & 0 & 0 \\ f_{21} & 1 & 0 \\ f_{31} & f_{32} & 1 \end{bmatrix} \tag{3.9}$$

LU decomposition algorithm:

- The factors generated during the elimination phase are stored in the lower part of the matrix
- Keep track of pivoting
- Scaled values of the elements are used to determine whether pivoting is to be implemented
- The diagonal term is monitored during the pivoting phase to detect near-zero occurrences in order to flag singular systems.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

3.2.2 The Matrix Inverse

The inverse can be computed in a column-by column fashion by generating solutions tieh unit vector as the right-hand-side constants. The best way to implement such a calculation is with the LU decomposition algorithm and it is one of the great strengths of LU decomposition.

3.2.3 Error Analysis and System Condition

Determination of ill-conditioned system:

- Scale the matrix and invert the scaled matrix. If there are element of $[A]^{-1}$ that are several orders of magnitude greater than one, then the system is ill-conditioned.
- Mutiply the inverse by the original matrix and assess whether the result is close to the identity matrix. If not, it indicate ill-conditioning.
- Invert the inversed matrix and assess whether the result is sufficiently close to the original matrix. If not, it indicate that the system is ill-conditioned.

The indication of ill-conditioning with a single number

- Norm: a real-valued function that provide a measure of the size of multicomponent mathematical entities. For example, consider a vector in three-dimensional Euclidean space

$$\begin{bmatrix} F \end{bmatrix} = \begin{bmatrix} a & b & c \end{bmatrix} \tag{3.10}$$

The length of this vector-that is, the distance from the coordinate (0, 0, 0) to (a, b, c)

$$\|F\|_e = \sqrt{a^2 + b^2 + c^2} \tag{3.11}$$

where the nomenclature $\|F\|_e$ indicates that this length is referred to as the Euclidean norm of $[F]$. For matrix case,

$$\|A\|_e = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2} \tag{3.12}$$

which is given a special name-the Frobenius norm. it provide a single value to quantify the "size" of $[A]$.

- Matrix condition number:

$$\mathrm{Cond}[A] = \|A\| \cdot \|A^{-1}\| \tag{3.13}$$

Note that for a matrix $[A]$, this number will be greater than or equal to 1. However, the above equation require computation time to obtain $\|A^{-1}\|$.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

3.3 Special Matrices and Gauss-Seidel

3.3.1 Special Matrices

- Banded matrix : a square matrix that has all elements equal to zero, with exception of a band centered on the main diagonal. The convensional LU decompostion methods are inefficient to solve banded systems.
- Tridiagonal system : Thomas algorithm
- Symmetric matrix : Cholesky decomposition

3.3.2 Gauss-Seidel

The Gauss-Seidel method:

- An alternative to the elimination method
- Iterative or approximate method

Convergence enhancement with relaxation

- Relaxation is a weighted average of the result of the previous and the present iteration:

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1-\lambda)x_i^{\text{old}} \tag{3.14}$$

- $\lambda = 1$: the result is unmodified.
- $0 < \lambda < 1$: underrelaxation, make a nonconvergent system converge or hasten convergence by dampening out oscillation.
- $\lambda > 1$: overrelaxation, accelerate the convergence of an already convergent system. It is also called successive or simultaneous overrelaxation, SOR.

3.3.3 Linear Algebraic Equation with Libraries and Packages

- Matlab:
  - cond : matrix condition number
  - norm : matrix or vector norm

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

  - rank : no. of linearly independent rows or columns
  - det : determinant
  - trace : sum of diagonal elements
  - / : linear equation solution
  - chol : cholesky factorization
  - lu : factors from gauss elimination
  - inv : matrix inverse

- IMSL: various routines are exist to solve linear system

3.4 Engineering Applications: Linear Algebraic Equations

See the textbook

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 4

Optimization

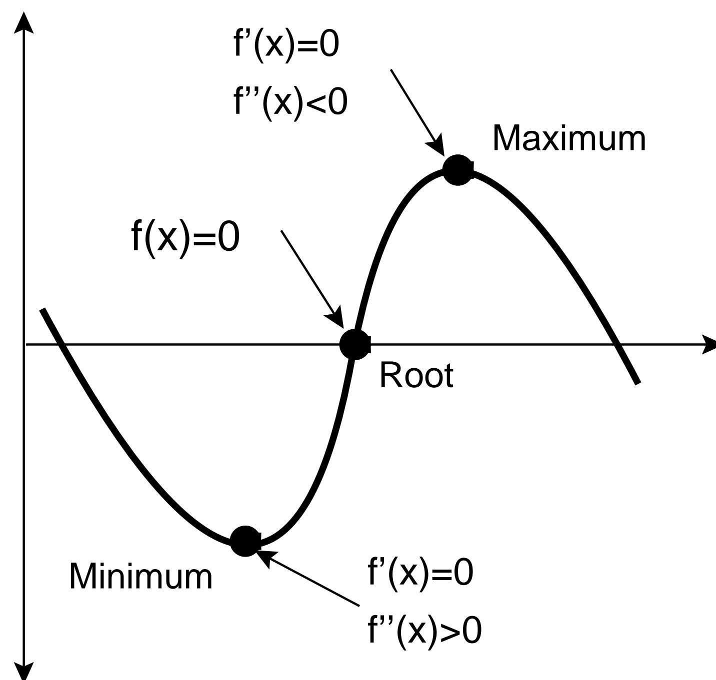

Figure 4.1: The illustation of the difference between roots and optima.

An optimization or mathematical programming problem

Find $\mathbf{x}$, which minimizes or maximizes $f(\mathbf{x})$

subject to

$$d_i(\mathbf{x}) \le a_i \quad i = 1, 2, \ldots, m \tag{4.1}$$
$$e_i(\mathbf{x}) = b_i \quad i = 1, 2, \ldots, p \tag{4.2}$$

where $\mathbf{x}$ is an n-dimensional design vector, $f(\mathbf{x})$ is the objective function, $d_i(\mathbf{x})$ are inequality constraints, $e_i(\mathbf{x})$ are equality constraints.

Classification of optimization problem

- The form of $f(\mathbf{x})$:
  - If $f(\mathbf{x})$ and the constraints are linear, *linear programming*.
  - If $f(\mathbf{x})$ is quadratic and the constraints are linear, *quadratic programming*.
  - If $f(\mathbf{x})$ is not linear or quadratic and/or the constraints are nonlinear, *nonlinear programming*.
- For constrained problem
  - Unconstained optimization
  - Constrained optimization
- Dimensionality
  - One-dimensional problem
  - Multi-dimensional problem

4.1 One-dimensional Unconstrained Optimization

4.1.1 Golden-Section Search

Golden-section search method is similar to the bisection method in solving for the root of a single nonlinear equation. Golden-section search method can be achived by specifying that the following two conditions hold:

$$\ell_0 = \ell_1 + \ell_2 \tag{4.3}$$
$$\frac{\ell_1}{\ell_0} = \frac{\ell_2}{\ell_1} \tag{4.4}$$

Defining $R = \ell_2/\ell_1$

$$R = 0.61803.... \tag{4.5}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

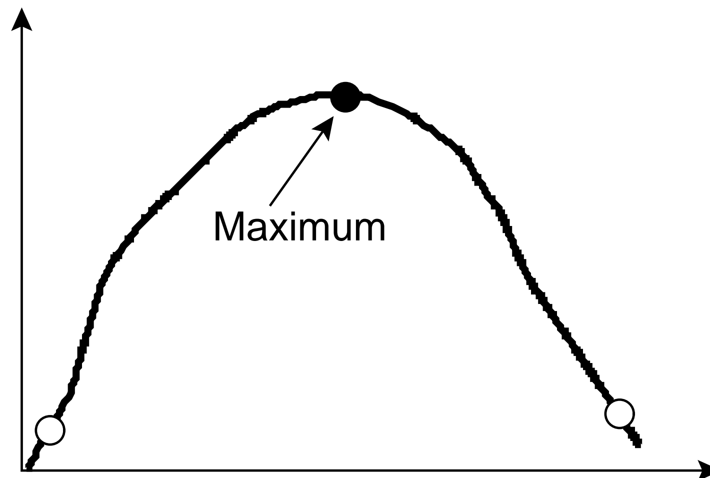

Figure 4.2: The illustation of the Golden-section search method.

This value is called the *golden ratio*.

Disadvantages

- Many evaluation
- Time-consuming evaluation

4.1.2 Quadratic Interpolation

Quadratic interpolation takes advantages of the fact that a second-order polynomial often provides a good approximation to the shape of $f(x)$ near an optimum.

An estimate of the optimal x

$$x_3 = \frac{f(x_0)(x_1^2 - x_2^2) + f(x_1)(x_2^2 - x_0^2) + f(x_2)(x_0^2 - x_1^2)}{2f(x_0)(x_1 - x_2) + 2f(x_1)(x_2 - x_0) + 2f(x_2)(x_0 - x_1)} \tag{4.6}$$

4.1.3 Newton's Method

At an optimum, the optimal value x^* satisfy

$$f'(x^*) = 0 \tag{4.7}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

With a second-order Taylor series of $f(x)$, we can find the following equations for an estimate of the optimal

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} \tag{4.8}$$

4.2 Multidimensional Unconstrained Optimization

Classification of unconstrained optimization problems

- Nongradient or direct methods
- Gradient or descent methods

4.2.1 Direct Methods

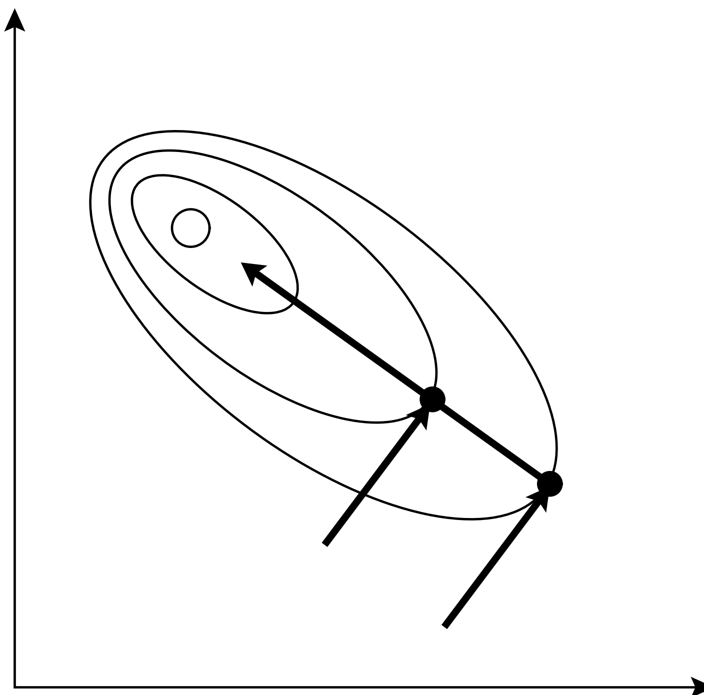

Figure 4.3: Conjugate directions.

These methods vary from simple brute force approaches to more elegant techniques that attempt to exploit the nature of the function.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- random search : repeatedly evaluates the function at randomly selected values of the independent variables.
- univariate search : change one variable at a time to improve the approximation while the other variables are held constant. Since only one variable is changed, the problem reduces to a sequence of one-dimensional searches.

4.2.2 Gradient Methods

Gradient methods use derivative information to generate efficient algorithms to locate optima.

The gradient is defined as

$$\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} \tag{4.9}$$

Derivative information

- First derivative:
  - a steepest trajectory of the function
  - whether it is a optima
- Second derivative: called as Hessian, H
  - If $|H| > 0$, it is a local minimum
  - If $|H| < 0$, it is a local maximum
  - If $|H| = 0$, it is a saddle point

The quantity $|H|$ is equal to the determinant of a matrix made up of the second derivatives and, for example, the Hessian of a two-dimensional system is

$$H = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

The steepest-descent algorithm is summaried as

- Determine the best direction
- Determine the best value along the search direction.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

1. Calculate the partial derivatives
$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_i} \tag{4.10}$$

2. Calculate the search vector
$$\mathbf{s} = -\nabla f(\mathbf{x}^k) \tag{4.11}$$

3. Use the relation
$$\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \mathbf{s}^k \tag{4.12}$$
to obtain the value of $\mathbf{x}^{k+1}$. To get λ^k use the following equations
$$f(\mathbf{x}^{k+1}) = f(\mathbf{x}^k + \lambda \mathbf{x}^k) = f(\mathbf{x}^k) + \nabla^T f(\mathbf{x}^k)\lambda \mathbf{s}^k + \frac{1}{2}(\lambda \mathbf{s}^k)^T \mathbf{H}(\mathbf{x}^k)(\lambda \mathbf{s}^k) \tag{4.13}$$
To get the minimum, differentiate with respect to λ and equate the derivative to zero
$$\frac{df(\mathbf{x}^k + \lambda \mathbf{x}^k)}{d\lambda} = \nabla^T f(\mathbf{x}^k)\mathbf{s}^k + (\mathbf{s}^k)^T \mathbf{H}(\mathbf{x}^k)(\lambda \mathbf{s}^k) \tag{4.14}$$
with the result
$$\lambda^{\text{opt}} = \frac{\nabla^T f(\mathbf{x}^k)\mathbf{s}^k}{(\mathbf{s}^k)^T \mathbf{H}(\mathbf{x}^k)\mathbf{s}^k} \tag{4.15}$$

4.3 Contrained Optimization

4.3.1 Linear Programming

Four general outcome from linear programming

- Unique solution
- Alternate solutions
- No feasible solution
- Unbounded problems

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

4.3.2 Optimization with Packages

- Matlab:
  - fmin : Minimize function of one variable
  - fmins : Minimiza function of several varaibles
  - fsolve : Solve nonlinear equations by a least squares method
- IMSL : various routines are exist to solve optimization problems

4.4 Engineering Applications: Optimization

See the textbook

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 5

Curve Fitting

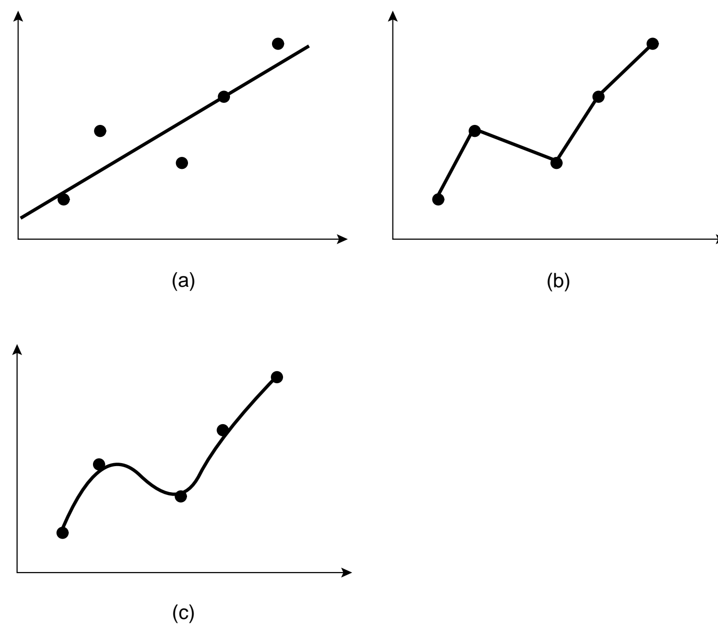

Figure 5.1: Three attempts to fit a best curve.

The simplest method for fitting a curve to data is to plot the points and then sketch a line

- (a) Characterize the general upward trend of the data with a straight line
- (b) Use straight-line segment or linear interpolation
- (c) Use curves to try to captuer the meanderings

Simple statistics

- Arithmetic mean

$$\bar{y} = \frac{\sum y_i}{n}$$

- Standard deviation : the measure of spread of a sample

$$s_y = \sqrt{\frac{S_t}{n-1}}$$

where S_t is the total sum of the squares of the residual between the data points and the mean, or

$$S_t = \sum (y_i - \bar{y})^2$$

- Variance : The square of the standard deviation

$$s_y^2 = \frac{S_t}{n-1}$$

- Coefficient of variation (c.v.) : The spread of data

$$c.v. = \frac{s_y}{\bar{y}} 100\%$$

5.1 Least-Squares Regression

Lest-squares regression is drived from a curve that minimized the discrepancy between the data points and the curve.

5.1.1 Linear Regression

A least-squares approximation is fitting a straight line to a set of paired observation. The mathematical expression for the straight line is

$$y = a_0 + a_1 x + e \tag{5.1}$$

The error, or residual, is the discrepancy between the true value of y and the approximate value, $a_0 + a_1 x$ and that is

$$e = y - a_0 + a_1 x \tag{5.2}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

The criterion for least-squares regression is

$$\min S_r = \sum_i^n e_i^2 = \sum_i^n (y_{i,\text{measured}} - y_{i,\text{model}})^2 = \sum_i^n (y_i - a_0 - a_1 x_i)^2 \tag{5.3}$$

To determine values of a_0 and a_1, differentiate (5.3)

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_i) \tag{5.4}$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum \left[(y_i - a_0 - a_1 x_i) x_i\right] \tag{5.5}$$

And setting these derivatives equal to zero, we get the so-called normal equations

$$0 = \sum y_i - \sum a_0 - \sum a_1 x_i \tag{5.6}$$

$$0 = \sum y_i x_i - \sum a_0 x_i - \sum a_1 x_i^2 \tag{5.7}$$

The coefficients of a straight line are

$$a_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2} \tag{5.8}$$

$$a_0 = \bar{y} - a_1 \bar{x} \tag{5.9}$$

Quantification of error of linear regression

- The sum of the square of the residual
  - A sampled data system

$$S_t = \sum (y_i - \bar{y})^2 \tag{5.10}$$

  - A linear regressioned system

$$S_r = \sum (y_i - a_0 - a_1 x_i)^2 \tag{5.11}$$

- Standard deviation
  - A sampled data system

$$s_y = \sqrt{\frac{S_t}{n-1}} \tag{5.12}$$

s_y quantifies the spread around mean.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

– A linear regressioned system

$$s_{y/x} = \sqrt{\frac{S_r}{n-2}} \tag{5.13}$$

$s_{y/x}$ quantifies the spread around the regression line.

- The goodness of a fit

$$r^2 = \frac{S_t - S_r}{S_t} \tag{5.14}$$

where r^2 is called the coefficient of determination and r is the correlation coefficient.

See the figure 17.4 in the textbook

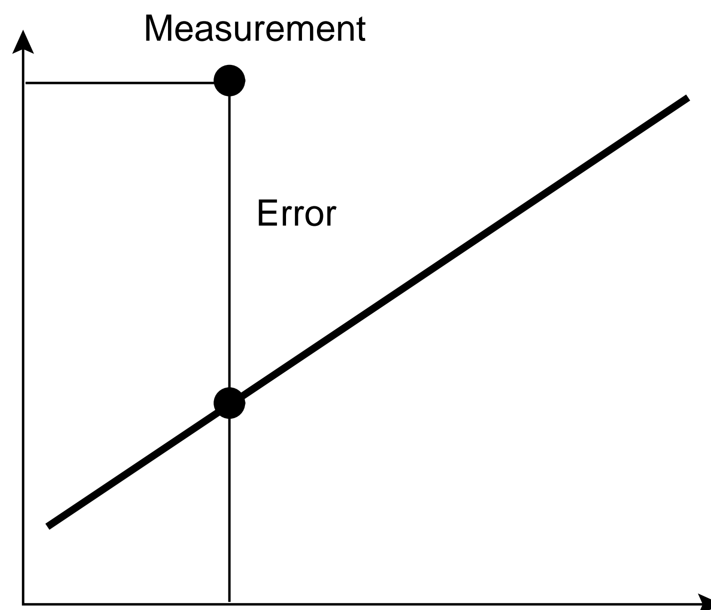

Figure 5.2: The residual in linear regression

5.1.2 General Linear Least-Squares

The general linear least-square model:

$$y = a_0 z_0 + a_1 z_1 + \cdots + a_m z_m + e \tag{5.15}$$

In matrix notation

$$Y = ZA + E \tag{5.16}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Note that Z is not a square matrix but we want to know about A.

$$Z^T Z A = Z^T Y \tag{5.17}$$

Now A is

$$A = (Z^T Z)^{-1} Z^T Y \tag{5.18}$$

5.1.3 Nonlinear Regression

Gauss-Newton method

1. Use a Taylor series to linearize a nonlinear function
2. Apply least-square theorie to obtain new estimate of the parameters that move in the direction of minimizing the residual.

5.2 Interpolation

5.2.1 Newton's Divided-Difference Interpolating Polynomials

Linear interpolation : connect two data points with a straight line

$$f_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) \tag{5.19}$$

Quadratic interpolation : connect three data points with a second-order polynomial

$$f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) \tag{5.20}$$

where

$$b_0 = f(x_0)$$
$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$
$$b_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Newton's interpolating polynomial : connect $n+1$ data with nth-order polynomial

$$f_n(x) = b_0 + b_1(x-x_0) + \cdots + b_n(x-x_0)\cdots(x-x_{n-1}) \tag{5.21}$$

where the coefficients are

$$\begin{aligned} b_0 &= f(x_0) \\ b_1 &= f[x_1, x_0] \\ &\vdots \\ b_n &= f[x_n, x_{n-1}, \ldots, x_0] \end{aligned}$$

where the bracket function evaluations are finite divided differences.

nth finite divided difference is

$$f[x_n, x_{n-1}, \ldots, x_0] = \frac{f[x_n, \ldots, x_1] - f[x_{n-1}, \ldots, x_0]}{x_n - x_0} \tag{5.22}$$

Newton's divided-difference interpolating polynomial is

$$f_n(x) = f(x_0) + (x-x_0)f[x_1, x_0] + \cdots (x-x_0)(x-x_1)\cdots(x-x_{n-1})f[x_n, \ldots, x_0] \tag{5.23}$$

5.2.2 Lagrange Interpolating Polynomial

The Lagrange interpolating polynomial is simply a reformulation of the Newton polynomial that avoids the computation of divided differences.

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i) \tag{5.24}$$

where

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{5.25}$$

where $\prod$ designates the "product of."

5.2.3 Spline Interpolation

Spline interpolation is an alternative approach that lower-order polynomial is applied to subsets of data point. Especially, when third-order curves are employed to connect each pair of data points, it is called cubic spline.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Linear splines : the simplest connection between two points is a straight line.

$$
\begin{aligned}
f(x) &= f(x_0) + m_0(x - x_0) && x_0 \le x \le x_1 \\
f(x) &= f(x_1) + m_1(x - x_1) && x_1 \le x \le x_2 \\
&\vdots \\
f(x) &= f(x_{n-1}) + m_1(x - x_{n-1}) && x_{n-1} \le x \le x_n
\end{aligned}
$$

where m_i is the slope of the straight line

$$m_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{5.26}$$

Quadratic splines : connect three points with second-order polynomials.

- The function values of adjacent polynomials must be equal at the interior knots.
- The first and last functions must pass through the end points.
- The first derivatives at the interior knots must be equal.
- Assume that the second derivative is zero at the first point.

Cubic splines : derive a third-order polynomial for each interval between knots

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \tag{5.27}$$

5.3 Fourier Approximation

In early 1800s, the French mathematician Fourier proposed that "any function can be represented by an infinite sum of sine and cosine terms." There are functions that do not have a representation as a Fourier series, however, most functions can be so represented. Fourier approximation is another representation of a function with trigonometric series.

Trigonometric identities

- $\sin A \sin B = \frac{1}{2}\left[\cos(A - B) - \cos(A + B)\right]$
- $\sin A \cos B = \frac{1}{2}\left[\sin(A - B) + \sin(A + B)\right]$
- $\cos A \cos B = \frac{1}{2}\left[\cos(A - B) + \cos(A + B)\right]$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Fourier series

Assume that $f(x)$ is a periodic function of period 2π and is integrable over a period.

$$f(x) \simeq A_0 + \sum_{n=1}^{\infty} [A_n \cos(nx) + B_n \sin(nx)] \tag{5.28}$$

- A_0 : integrating on both sides of (5.28) from $-\pi$ to π.

$$\int_{-\pi}^{\pi} f(x)dx = \int_{-\pi}^{\pi} A_0 dx + \sum_{n=1}^{\infty} A_n \int_{-\pi}^{\pi} \cos(nx)dx + \sum_{n=1}^{\infty} B_n \int_{-\pi}^{\pi} \sin(nx)dx \tag{5.29}$$

The last two integrations of trigonometric terms are equal to zero. Hence

$$A_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)dx \tag{5.30}$$

- A_n : multiply both sides of (5.28) by $\cos(mx)$ and integrate

$$\begin{aligned} \int_{-\pi}^{\pi} \cos(mx) f(x)dx &= \int_{-\pi}^{\pi} A_0 \cos(mx)dx \\ &+ \sum_{n=1}^{\infty} \int_{-\pi}^{\pi} A_n \cos(nx)\cos(mx)dx + \sum_{n=1}^{\infty} \int_{-\pi}^{\pi} B_n \sin(nx)\cos(mx)dx \end{aligned} \tag{5.31}$$

The only nonzero term on the right is when $m = n$ in the first summation

$$A_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\cos(nx)dx \tag{5.32}$$

- B_n : multiply both sides of (5.28) by $\sin(mx)$ and integrate

$$\begin{aligned} \int_{-\pi}^{\pi} \sin(mx) f(x)dx &= \int_{-\pi}^{\pi} A_0 \sin(mx)dx \\ &+ \sum_{n=1}^{\infty} \int_{-\pi}^{\pi} A_n \cos(nx)\sin(mx)dx + \sum_{n=1}^{\infty} \int_{-\pi}^{\pi} B_n \sin(nx)\sin(mx)dx \end{aligned} \tag{5.33}$$

The only nonzero term on the right is when $m = n$ in the second summation

$$B_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\sin(nx)dx \tag{5.34}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Fourier series for any period $p = 2L$

Consider the function whose period is $p = 2L$.

$$f(x) = A_0 + \sum_{n=1}^{\infty} \left(A_n \cos \frac{n\pi}{L} x + B_n \sin \frac{n\pi}{L} x \right) \tag{5.35}$$

where the Fourier coefficients of $f(x)$ are given by the Euler formulas

$$\begin{aligned} A_0 &= \frac{1}{2L} \int_{-L}^{L} f(x) dx \\ A_n &= \frac{1}{L} \int_{-L}^{L} f(x) \cos \frac{n\pi}{L} x dx \\ B_n &= \frac{1}{L} \int_{-L}^{L} f(x) \sin \frac{n\pi}{L} x dx \end{aligned} \tag{5.36}$$

Fourier series for even and odd functions

- Even function:

$$g(-x) = g(x) \tag{5.37}$$

  And integral value of a even function is

$$\int_{-L}^{L} g(x) dx = 2 \int_{0}^{L} g(x) dx \tag{5.38}$$

- Odd function:

$$h(-x) = -h(x) \tag{5.39}$$

  And integral value of a even function is

$$\int_{-L}^{L} h(x) dx = 0 \tag{5.40}$$

- Fourier cosine series: the Fourier series of an even function of period $2L$.

$$f(x) = A_0 + \sum_{n=1}^{\infty} A_n \cos \frac{n\pi}{L} x \tag{5.41}$$

- Fourier sine series: the Fourier series of an odd function of period $2L$.

$$f(x) = A_0 + \sum_{n=1}^{\infty} B_n \sin \frac{n\pi}{L} x \tag{5.42}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Complex form of Fourier series : Real sines and cosines can be expressed in terms of complex exponentials by the formulas

$$\sin nx = \frac{e^{inx} - e^{-inx}}{2i}$$
$$\cos nx = \frac{e^{inx} + e^{-inx}}{2} \tag{5.43}$$

From this

$$A_n \cos nx + B_n \sin nx = \frac{1}{2}A_n(e^{inx} + e^{-inx}) + \frac{1}{2i}B_n(e^{inx} - e^{-inx}) \tag{5.44}$$
$$= \frac{1}{2}(A_n - iB_n)e^{inx} + \frac{1}{2}(A_n + iB_n)e^{-inx} \tag{5.45}$$
$$= c_n e^{inx} + c_{-n}e^{-inx} \tag{5.46}$$

With the above equation

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{inx} \tag{5.47}$$

where

$$c_n = A_n - iB_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)(\cos(nx) - i\sin(nx))dx = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx$$

This is the so-called complex form of the Fourier series, or complex Fourier series of $f(x)$.

Sinusoidal function : represent any waveform with a sine or cosine

$$f(t) = A_0 + C_1\cos(\omega_0 t + \theta) \tag{5.48}$$

where A_0 is the mean value, C_1 is the amplitude, ω_0 is the angular frequency, and θ is the phase angle or phase shift.

The angular frequency is related to frequency f (in cycles/time)

$$\omega_0 = 2\pi f \tag{5.49}$$

and frequency is

$$f = \frac{1}{T} \tag{5.50}$$

The trigonometric identity gives

$$f(t) = A_0 + A_1\cos(\omega_0 t) + B_1\sin(\omega_0 t) \tag{5.51}$$

where $A_1 = C_1\cos(\theta)$, $B_1 = -C_1\sin(\theta)$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

5.3.1 Curve Fitting with Sinusoidal Functions

Least-squares fit of a sinusoidal function is to determine coefficient values that minimize

$$S_r = \sum_{i=1}^{N} \{y_i - [A_0 + A_1 \cos(\omega_0 t_i) + B_1 \sin(\omega_0 t_i)]\}^2 \tag{5.52}$$

$$\begin{bmatrix} N & \sum \cos(\omega_0 t) & \sum \sin(\omega_0 t) \\ \sum \cos(\omega_0 t) & \sum \cos^2(\omega_0 t) & \sum \cos(\omega_0 t)\sin(\omega_0 t) \\ \sum \sin(\omega_0 t) & \sum \cos(\omega_0 t)\sin(\omega_0 t) & \sum \sin^2(\omega_0 t) \end{bmatrix} \begin{Bmatrix} A_0 \\ A_1 \\ B_1 \end{Bmatrix} = \begin{Bmatrix} \sum y \\ \sum y\cos(\omega_0 t) \\ \sum y \sin(\omega_0 t) \end{Bmatrix} \tag{5.53}$$

For equispaced system

$$\int_0^T \cos(\omega_0 t) dt = -\frac{1}{\omega_0} \sin(\omega_0 t)\Big|_0^T = 0 \tag{5.54}$$

where $\omega_0 T = \frac{2\pi}{T} T = 2\pi$. These relationhips give

$$\begin{bmatrix} N & 0 & 0 \\ 0 & N/2 & 0 \\ 0 & 0 & N/2 \end{bmatrix} \begin{Bmatrix} A_0 \\ A_1 \\ B_1 \end{Bmatrix} = \begin{Bmatrix} \sum y \\ \sum y\cos(\omega_0 t) \\ \sum y \sin(\omega_0 t) \end{Bmatrix} \tag{5.55}$$

or

$$A_0 = \frac{\sum y}{N} \tag{5.56}$$

$$A_1 = \frac{2}{N} \sum y \cos(\omega_0 t) \tag{5.57}$$

$$A_2 = \frac{2}{N} \sum y \sin(\omega_0 t) \tag{5.58}$$

The above equations are similar with the determination of Fourier series.

5.3.2 Fourier Integral and Transform

Some of phenomenon does not occured repeatedly or it will be a long time until it occurs again. In this case we use Fourier integral that can be used to represent nonperiodic functions, for example a single voltage pulse not repeated, or a flash of light, or a sound which is not repeated. The transition from a periodic to a nonperiodic function can be effected by allowing the period to approach infinity. In other words, as T becomes infinite, the function never repeats itself and thus becomes aperiodic.

From Fourier series to the Fourier intergral

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

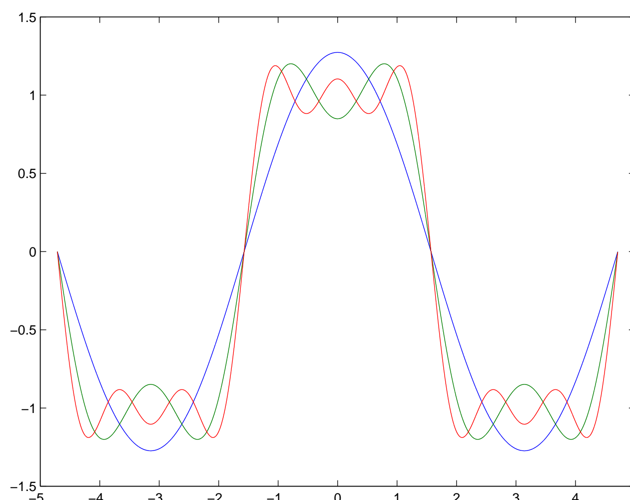

Figure 5.3: The Fourier series approximation of the square wave.

Consider any periodic function $f_L(x)$ of period $2L$

$$f_L(x) = A_0 + \sum_{n=1}^{\infty}(A_n \cos \omega_n x + B_n \sin \omega_n x) \tag{5.59}$$

where $\omega_n = n\pi/L$. Insert A_n and B_n which are given by the Euler formulas.

$$\begin{aligned} f_L(x) = &\frac{1}{2L}\int_{-L}^{L} f_L(v)dv \\ &+ \frac{1}{L}\sum_{n=1}^{\infty}\left[\cos \omega_n x \int_{-L}^{L} f_L(v)\cos \omega_n v dv + \sin \omega_n x \int_{-L}^{L} f_L(v)\sin \omega_n v dv\right] \end{aligned} \tag{5.60}$$

Now set

$$\Delta\omega = \omega_{n+1} - \omega_n = \frac{(n+1)\pi}{L} - \frac{n\pi}{L} = \frac{\pi}{L} \tag{5.61}$$

Then $1/L = \Delta\omega/\pi$, and

$$\begin{aligned} f_L(x) = &\frac{1}{2L}\int_{-L}^{L} f_L(v)dv \\ &+ \frac{1}{\pi}\sum_{n=1}^{\infty}\left[\cos \omega_n x\Delta\omega \int_{-L}^{L} f_L(v)\cos \omega_n v dv + \sin \omega_n x\Delta\omega \int_{-L}^{L} f_L(v)\sin \omega_n v dv\right] \end{aligned} \tag{5.62}$$

Let $L \longrightarrow \infty$ and assume a periodic function $f_L(x)$ to be a aperiodic function.

$$f(x) = \lim_{L\to\infty} f_L(x) \tag{5.63}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Then $1/L \longrightarrow 0$ and the first term of function approaches zero.

$$f_L(x) = \frac{1}{\pi}\sum_{n=1}^{\infty}\left[\cos\omega_n x\Delta\omega\int_{-L}^{L} f_L(v)\cos\omega_n v dv + \sin\omega_n x\Delta\omega\int_{-L}^{L} f_L(v)\sin\omega_n v dv\right] \quad (5.64)$$

$L \longrightarrow 0$ results in $\Delta\omega \to 0$ and the sum of infinite series become an integral from 0 to ∞.

$$f(x) = \frac{1}{\pi}\int_0^{\infty}\left[\cos\omega x\int_{-\infty}^{\infty} f(v)\cos\omega v dv + \sin\omega x\int_{-\infty}^{\infty} f(v)\sin\omega v dv\right] d\omega \quad (5.65)$$

Introduce $A(\omega)$ and $B(\omega)$ as

$$A(\omega) = \int_{-\infty}^{\infty} f(v)\cos\omega v dv, \quad B(\omega) = \int_{-\infty}^{\infty} f(v)\sin\omega v dv \quad (5.66)$$

Finally Fourier series for an aperiodic equation become

$$f(x) = \int_0^{\infty}\left[A(\omega)\cos\omega x + B(\omega)\sin\omega x\right] d\omega \quad (5.67)$$

This is called a representation of $f(x)$ by a Fourier integral.

Alternatively, the Fourier integral can be written as complex Fourier series.

$$\begin{aligned} f(x) &= \sum_{-\infty}^{\infty} c_n e^{i\omega_n x} \\ c_n &= \frac{1}{2L}\int_{-L}^{L} f(u)e^{-i\omega_n u} du \end{aligned} \quad (5.68)$$

$$f(x) = \sum_{-\infty}^{\infty}\left[\frac{1}{2L}\int_{-L}^{L} f(u)e^{-i\omega_n u} du\right] e^{i\omega_n x} \quad (5.69)$$

Use $1/L = \Delta\omega/\pi$

$$f(x) = \sum_{-\infty}^{\infty}\left[\frac{\Delta\omega}{2\pi}\int_{-L}^{L} f(u)e^{-i\omega_n u} du\right] e^{inx} \quad (5.70)$$

$$= \sum_{-\infty}^{\infty}\frac{\Delta\omega}{2\pi}\int_{-L}^{L} f(u)e^{i\omega_n(x-u)} du = \frac{1}{2\pi}\sum_{-\infty}^{\infty} F(\omega_n)\Delta\omega \quad (5.71)$$

where

$$F(\omega_n) = \int_{-L}^{L} f(u)e^{i\omega_n(x-u)} du \quad (5.72)$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

If $\Delta\omega$ goes to zero, a limit of a sum becomes an integral

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)d\omega = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u)e^{i\omega(x-u)}dud\omega \tag{5.73}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x}d\omega \int_{-\infty}^{\infty} f(u)e^{-i\omega u}du \tag{5.74}$$

Define $g(\omega)$ by

$$g(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(u)e^{-i\omega u}du \tag{5.75}$$

Then

$$f(x) = \int_{-\infty}^{\infty} g(\omega)e^{i\omega x}d\omega \tag{5.76}$$

Fourier Transform

$$\begin{aligned} f(x) &= \int_{-\infty}^{\infty} g(\omega)e^{i\omega x}d\omega \\ g(\omega) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} f(u)e^{-i\omega u}du \end{aligned} \tag{5.77}$$

$f(x)$ and $g(\omega)$ are called a pair of Fourier transforms. Usually, $g(\omega)$ is called the Fourier transform of $f(x)$, and $f(x)$ is called the inverse Fourier transform of $g(\omega)$.

5.3.3 Discrete Fourier Transform (DFT)

In engineering, functions are often represented by finite sets of discrete values and data is often collected in or converted to such a discrete format. For the discrete time system, a discrete Fourier transform can be written as

$$F_k = \sum_{n=0}^{N-1} f_n e^{-i\omega_0 n} \tag{5.78}$$

and the inverse Fourier transform as

$$f_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k e^{i\omega_0 n} \tag{5.79}$$

where $\omega_0 = 2\pi/N$.

5.3.4 Fast Fourier Transform (FFT)

The fast Fourier transform (FFT) is an algorithm that has been developed to compute the DFT in an extremely economical fashion.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

5.3.5 The Power Spectrum

A power spectrum is developed from the Fourier transform and it is derived from the analysis of the power output of electrical systems. The power of a periodic signal can be defined as

$$P = \frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt \tag{5.80}$$

A power spectrum can be calculated by the power associated with each frequency component.

5.3.6 Curve Fitting with Libraries and Packagies

- Matlab:
  - polyfit
  - polyval
  - poly2sym
  - interp1
  - spline
  - fft
- IMSL: various routines are exist to solve curve fitting and fft problems

5.4 Engineering Applications: Curve Fitting

See the textbook

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 6

Numerical Differentiation and Integration

The derivative represents the rate of cchange of a dependent variable with respect to an independent variable.

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} \tag{6.1}$$

The integration means the total value, or summation, of $f(x)dx$ over the range $x = a$ to b.

$$I = \int_b^a f(x)dx \tag{6.2}$$

6.1 Newton-Cotes Integration of Equations

Newton-Gregory forward polynomial : If the x-values are evenly spaced, instead of using divided difference, "ordinary differences" are more useful; the differences in f-values are not divided by the differences in x-values.

$$\begin{aligned} P_n(x_s) =& f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3}\Delta^3 f_0 + \cdots \\ &+ \frac{s(s-1)\cdots(s-n+1)}{n}\Delta^n f_0 \end{aligned} \tag{6.3}$$

where $s = (x - x_0)/h$, with $h = \Delta x$, the uniform spacing in x-values.

The Newton-Cotes formulas are based on the strategy of replacing a complicated function or tabulated data with an approximating function that is easy to integrate:

$$I = \int_b^a f(x)dx \simeq \int_b^a P_n(x)dx \tag{6.4}$$

where $P_n(x)$ is the Newton-Gregory interpolating polynomial. For $n = 1$

$$\begin{aligned}\int_b^a f(x)dx \simeq \int_b^a P_1(x)dx = \int_b^a (f_0 + s\Delta f_0)dx = h\int_0^1 (f_0 + s\Delta f_0)ds \\ = h\left(f_0 + \frac{1}{2}\Delta f_0\right) = \frac{h}{2}(f_0 + f_1)\end{aligned} \tag{6.5}$$

For $n = 2$

$$\begin{aligned}\int_b^a f(x)dx &\simeq \int_b^a \left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0\right)dx \\ &= h\int_0^1 \left(f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0\right)ds \\ &= \frac{h}{3}(f_0 + 4f_1 + f_2)\end{aligned} \tag{6.6}$$

See the figure 21.1 in the textbook.

6.1.1 The Trapezoidal rule

The trapezoidal rule is the first of the Newton-Cotes closed integration formulas

$$I = \int_b^a f(x)dx \simeq \int_b^a f_1(x)dx \tag{6.7}$$

where

$$f_1(x) = f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \tag{6.8}$$

The result of integration is

$$I = (b - a)\frac{f(b) - f(a)}{2} \tag{6.9}$$

which is called as trapezoidal rule.

One way to improve the accuracy of the trapezoidal rule is to divide the integration interval from a to b into a number of segments and apply the method to each segment. The width of segments

$$h = \frac{b - a}{n} \tag{6.10}$$

The integration is

$$I = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right] \tag{6.11}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

6.1.2 Simpson's rule

Another way to obtain a more accurate estimate of an integral is to use higher-order polynomial to connect the points.

Simpson's 1/3 rule : use a second-order polynomial

$$I = \int_a^b f(x)dx \simeq \int_a^b f_2(x)dx \tag{6.12}$$

Simpson's 1/3 rule is

$$I \simeq \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] \tag{6.13}$$

The label "1/3" stems from the fact that h is divided by 3.

Simpson's 3/8 rule : use a third-order Lagrange polynomial

$$I = \int_a^b f(x)dx \simeq \int_a^b f_3(x)dx \tag{6.14}$$

Simpson's 3/8 rule is

$$I \simeq \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] \tag{6.15}$$

See the figure 21.11 in the textbook.

6.2 Intergrations of Equations

6.2.1 Romberg integration

Richardson's extrapolation : use two estimates of an integral to compute a third. It improves the results of numerical integration on the basis of the integral estimate themselves.

Two separate estimate using step sizes of h_1 and h_2

$$I(h_1) + E(h_1) = I(h_2) + E(h_1) \tag{6.16}$$

The error of the multiple-application trapezoidal rule is

$$E \simeq -\frac{b-a}{12}h^2\bar{f}'' \tag{6.17}$$

Assume that $\bar{f}''$ is constant regardless of step size

$$\frac{E(h_1)}{E(h_2)} \simeq \frac{h_1^2}{h_2^2} \tag{6.18}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Rearranage the above equation

$$E(h_1) \simeq E(h_2)\left(\frac{h_1}{h_2}\right)^2 \tag{6.19}$$

which can be substituted into eq. (6.16)

$$I(h_1) + E(h_2)\left(\frac{h_1}{h_2}\right)^2 \simeq I(h_2) + E(h_1) \tag{6.20}$$

which can be solved for

$$E(h_2) \simeq \frac{I(h_1) - I(h_2)}{1 - (h_1/h_2)^2} \tag{6.21}$$

Thus, we have developed an estimate of the truncation error in terms of the integral estimates and their step sizes. This estimate can then be substituted into

$$I = I(h_2) + E(h_2) \tag{6.22}$$

to yield an improved estimate of the integral:

$$I \simeq I(h_2) + \frac{1}{(h_1/h_2)^2 - 1}\left[I(h_2) - I(h_1)\right] \tag{6.23}$$

For the special case where the interval is halved $(h_2 = h_1/2)$

$$I \simeq I(h_2) + \frac{1}{2^2 - 1}\left[I(h_2) - I(h_1)\right] \tag{6.24}$$

or

$$I \simeq \frac{4}{3}I(h_2) - \frac{1}{3}I(h_1) \tag{6.25}$$

The Romberg integration algorithm

$$I_{j,k} \simeq \frac{4^{k-1}I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1} \tag{6.26}$$

where $I_{j+1,k+1}$ and $I_{j,k-1}$ are the more and less accurate integral and Ij, k is the improved integral.

6.2.2 Gauss Quadrature

- Trapezoidal rule : two parameters model

  $$I \simeq c_0 f(a) + c_1 f(b) \tag{6.27}$$

  where the c's are the unknown parameters.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- Gauss Quadrature : four parameters model

$$I \simeq c_0 f(a) + c_1 f(b) \tag{6.28}$$

where the c's, $f(a)$, $f(b)$ are the unknown parameters.

The trapezoidal rule's formula can be derived from another point of view, the method of undetermined coefficients. Because the trapezoidal rule is a two parameters model, we need two relationships that connect two parameters.

$$c_0 + c_1 = \int_{-(b-a)/2}^{(b-a)/2} 1dx \tag{6.29}$$

and

$$-c_0 \frac{b-a}{2} + c_1 \frac{b-a}{2} = \int_{-(b-a)/2}^{(b-a)/2} xdx \tag{6.30}$$

$$I \simeq (b-a) \frac{f(a) + f(b)}{2} \tag{6.31}$$

The trapezoidal rule must pass through the end point and results in a large error. But suppose that the constraints of fixed base points was removed and we were freely evaluate the area under a straight line joining any two points on the curve. See the figure 22.5 to figure out the differences.

The object of Gauss quadrature is to determine the coefficients of an equation of the form

$$I \simeq c_0 f(a) + c_1 f(b) \tag{6.32}$$

with assuming that eq. (6.32) fit the integral of a constant, a linear, a parabolic, and a cubic function

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} dx = 2 \tag{6.33}$$

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} xdx = 0 \tag{6.34}$$

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} x^2 dx = \frac{2}{3} \tag{6.35}$$

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} x^3 dx = 0 \tag{6.36}$$

These relationships yield the two-point Gauss-Legendre formula

$$I \simeq f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}}) \tag{6.37}$$

Because Gauss quadrature requires function evalutions at nonuniformly spaced points within the integration interval, it is not appropriate for cases where the function is unknown.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

6.2.3 Improper integrals

Improper intergral that is one with a lower limit of $-\infty$ or an upper limit of $+\infty$, usually can be evaluated by makeing a change of variable that transforms the infinite range to one that is finite.

6.3 Numerical Differentiation

Improve derivative estimates

- Decrease the step size
- Use a higher-order formula
- Combine two derivative estimates to compute more accurate approximation

6.3.1 High-accuracy differentiation formulas

The forward Taylor series expansion can be written as

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2}h^2 + \cdots \tag{6.38}$$

which can be solved for

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f''(x_i)}{2}h^2 + O(h^2) \tag{6.39}$$

If we truncate the second- and higher-derivative terms

$$f'(x_i) = \frac{f(x_{i+1} - f(x_i)}{h} \tag{6.40}$$

The accuracy of the above equation depend on the step size h.

In contrast to this approach, substitue the second-derivative term

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h) \tag{6.41}$$

into eq. (6.39) to yield

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{2h^2}h + O(h^2) \tag{6.42}$$

or, by collecting terms,

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2) \tag{6.43}$$

Notice that inclusion of the second-derivative term has improved the accuracy to $O(h^2)$.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

6.3.2 Richardson extrapolation

In similar with Richardson extrapolation for integral, an estimate for derivatives can be written as

$$D \simeq \frac{4}{3}D(h_2) - \frac{1}{3}D(h_1) \tag{6.44}$$

For centered difference approximations with $O(h^2)$, the application of this formula will yield a new derivative estimate of $O(h^4)$.

6.3.3 Derivatives of unequally spaced data

One way to handle nonequispaceddata is to fit a second-order Lagrange interpolating polynomial to each set of three adjacent points.

6.3.4 Numerical integration/differentiation formulas with libraties and packages

Various subroutines and functions are exist to solve integral and derivative problems in Matlab and IMSL.

6.4 Engineering Applications: Numerical Integration and Differentiation

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 7

Ordinary Differential Equations

Differential equation is

- differential equations : composed of an unknown function and its derivatives.
- rate equations : it expresses the rate of change of a variable as a function of variables and parameters.

Variables are divied by

- dependent variables
- independent variables

Differential equation is classified as

- ordinary differential equation : one independent variable
- partial differential equation : more than one independent variables

A differential equation is usually accompanied by auxiliary conditions to specify the solution completely. For first-order ODEs an initial value is required to determine the constant and obtain a unique solution.

- initial-value problem : all conditions are specified at the same value of the independent variable.
- boundary-value problem : specification of conditions occurs at different values of the independent variable.

7.1 Runge-Kutta Methods

Ordinary differential equation is

$$\frac{dy}{dx} = f(x, y)$$

The solution is

$$\text{New value} = \text{old value} + \text{slope} \times \text{step size}$$

or, in mathmatical terms,

$$y_{i+1} = y_i + \phi \times h \tag{7.1}$$

The slope estimate of ϕ is used to extrapolate from an old value y_i to a new value y_{i+1} over a distance h.

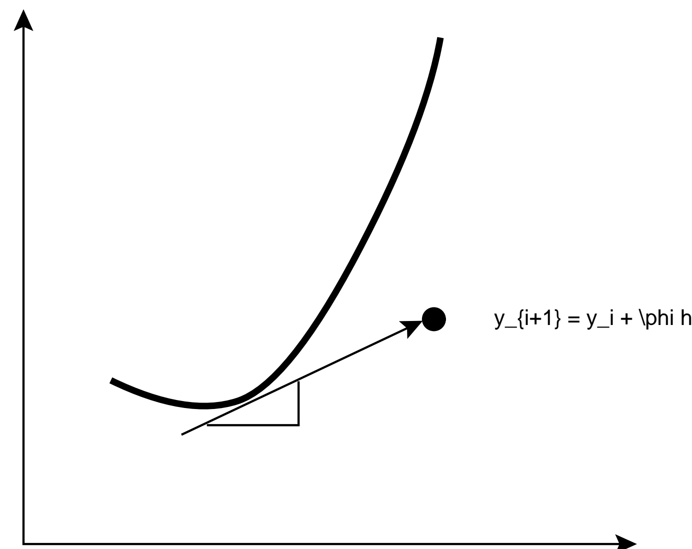

Figure 7.1: Graphical depiction of a one-step method.

7.1.1 Euler's Method

Euler's method is

$$y_{i+1} = y_i + f(x_i, y_i)h \tag{7.2}$$

More smaller step-size gives more accurate solution but further step-size reduction requires much computation times.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

7.1.2 Improvement of Euler's Method

A funcdermental source of error in Euler's method is that the derivative at the beginning of the interval is assumed to apply across the entire interval.

Heun's Method : average two derivatives which are obtained at the initial point and the end point. The slope at the beginning of an interval

$$y_i' = f(x_i, y_i) \tag{7.3}$$

is used to extrapolate linearly to y_{i+1}

$$y_{i+1}^0 = y_i + f(x_i, y_i)h \tag{7.4}$$

This equation is called a predictor equation. The slope at the end of the interval

$$y_{i+1}' = f(x_{i+1}, y_{i+1}) \tag{7.5}$$

Thus, the two slopes can be combined to obtain an average slope for the interval

$$\bar{y}' = \frac{y_i + y_{i+1}'}{2} = \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} \tag{7.6}$$

This average slope is then used to extrapolate linearly from y_i to y_{i+1}

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2}h \tag{7.7}$$

which is called a correct equation. See the figure 25.9 at p 688.

The Heun method is a predictor-corrector approach.

The Midpoint Method : use Euler's method to predict a value of y at the midpoint of the interval.

$$y_{i+1/2} = y_i + f(x_i, y_i)\frac{h}{2} \tag{7.8}$$

This slope is then used to extrapolate linear form from x_i to x_{i+1}

$$y_{i+1} = y_i + f(x_{i+1/2}, y_{i+1/2})\frac{h}{2} \tag{7.9}$$

7.1.3 Runge-Kutta Method

Runge-Kutta methods achieve the accuracy of a Taylor series approach without requiring the calculation of higher derivatives.

$$y_{i+1} = y_i + \phi(x_i, y_i, h)h \tag{7.10}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

where $\phi(x_i, y_i, h)$ is called an increment function, which can be interpreted as a representative slope over the interval. The increment function is

$$\phi = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n \tag{7.11}$$

where the a's are constants and the k's are

$$k_1 = f(x_i, y_i) \tag{7.12a}$$
$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h) \tag{7.12b}$$
$$k_3 = f(x_i + p_2 h, y_i + q_{21} k_1 h + q_{22} k_2 h) \tag{7.12c}$$
$$\vdots \tag{7.12d}$$
$$k_n = f(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 h + \cdots + q_{n-1,n-1} k_{n-1} h) \tag{7.12e}$$

Notice that the k's are recurrence relationship. Because each k is a functional evaluation, this recurrence makes RK methods efficient for computer calculations.

Second-order Runge-Kutta Methods

The second-order version of RK method:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2) h \tag{7.13}$$

where

$$k_1 = f(x_i, y_i) \tag{7.14a}$$
$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h) \tag{7.14b}$$

To determine values for the constant a_1, a_2, p_1, and q_{11}, use a Taylor's series for y_{i+1} in terms of y_i and $f(x_i, y_i)$

$$y_{i+1} = y_i + f(x_i, y_i) h + \frac{f'(x_i, y_i)}{2!} h^2 \tag{7.15}$$

where $f'(x_i, y_i)$ is

$$f'(x_i, y_i) = \frac{\partial f(x, y)}{\partial x} + \frac{\partial f(x, y)}{\partial y} \frac{dy}{dx} \tag{7.16}$$

Then,

$$y_{i+1} = y_i + f(x_i, y_i) h + \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{dy}{dx} \frac{h^2}{2!} \tag{7.17}$$

The basic strategy underlying Runge-Kutta methods is to use algebraic manipulations to solve for values of a_1, a_2, p_1, and q_{11} that make eq (7.13) and eq (7.17) equivalent.

The Taylor's series for a two-variable function is

$$f(x_i + p_1 h, y_i + q_{11} k_1 h) = f(x_i, y_i) + p_1 h \frac{\partial f}{\partial x} + q_{11} k_1 h \frac{\partial f}{\partial y} + O(h^2) \tag{7.18}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

This gives

$$y_{i+1} = y_i + a_1 h f(x_i, y_i) + a_2 h f(x_i, y_i) + a_2 p_1 h^2 \frac{\partial f}{\partial x} + a_2 q_{11} h^2 f(x_i, y_i) \frac{\partial f}{\partial y} + O(h^3) \quad (7.19)$$

By correcting terms

$$y_{i+1} = y_i + [a_1 f(x_i, y_i) + a_2 f(x_i, y_i)]\, h + \left[a_2 p_1 \frac{\partial f}{\partial x} + a_2 q_{11} f(x_i, y_i) \frac{\partial f}{\partial y} \right] h^2 + O(h^3) \quad (7.20)$$

Comparing this equation with eq (7.17)

$$a_1 + a_2 = 1 \quad (7.21)$$

$$a_1 p_1 = \frac{1}{2} \quad (7.22)$$

$$a_2 q_{11} = \frac{1}{2} \quad (7.23)$$

Because these three equations contain the four unknown constants, we must assume a value of one of the unknowns to determine the other three. Suppose that we specify a value for a_2.

$$a_1 = 1 - a_2 \quad (7.24)$$

$$p_1 = q_{11} = \frac{1}{2a_2} \quad (7.25)$$

Also we can choose an infinite number of values for a_2, there are an infinite number of second-order RK methods.

Heun Method with a Single Corrector($a_2 = 1/2$)

Assume $a_2 = 1/2$

$$a_1 = 1/2, \quad p_1 = q_{11} = 1 \quad (7.26)$$

These parameters yield

$$y_{i+1} = y_i + \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right) h \quad (7.27)$$

where

$$k_1 = f(x_i, y_i) \quad (7.28)$$

$$k_2 = f(x_i + h, y_i + k_1 h) \quad (7.29)$$

Midpoint Method($a_2 = 1$)

$$y_{i+1} = y_i + k_2 h \quad (7.30)$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

where

$$k_1 = f(x_i, y_i) \tag{7.31}$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \tag{7.32}$$

Ralston's Method ($a_2 = 2/3$)

$$y_{i+1} = y_i + \left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)h \tag{7.33}$$

where

$$k_1 = f(x_i, y_i) \tag{7.34}$$

$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right) \tag{7.35}$$

See the figure 25.14 at p 699.

Fourth-order Runge-Kutta Methods

The classical fourth-order RK method

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h \tag{7.36}$$

where

$$k_1 = f(x_i, y_i) \tag{7.37a}$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \tag{7.37b}$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \tag{7.37c}$$

$$k_4 = f\left(x_i + h, y_i + k_3 h\right) \tag{7.37d}$$

(7.37e)

See the figure 25.16 at p 704.

7.1.4 Systems of Equations

The procedure for solving a system of equations simply involves applying the one-step technique for every equation at each step before proceeding to the next step.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

7.1.5 Adaptive Runge-Kutta Method

- Automatically adjust the step size to avoid overkills
- Need an estimate of the local truncation error be obtained at each step

Strategies for adaptive RK

- Use different step sizes to calculate local error
- Use different order of RK method to calculate local error

7.2 Stiffness and Multistep Methods

7.2.1 Stiffness

A stiffness system is one involving repidly changing components together with slowly changing ones. In many cases, the rapidly varying components die away quickly, after which the solution becomes dominated by the slowly varying components.

An example of a single stiff ODE is

$$\frac{dy}{dt} = -1000y + 3000 - 2000e^{-t} \tag{7.38}$$

If $y(0) = 0$,

$$y = 3 - 0.998e^{-1000t} - 2.002e^{-t} \tag{7.39}$$

The solution is initially dominated by the fast exponential term (e^{-1000t}). After a very short period $t < 0.005$, this transient dies out and the solution becomes dictated by the slow exponential (e^{-t}).

Step size consideration: Insight into the step size required for stability for a solution.

$$\frac{dy}{dt} = -ay \tag{7.40}$$

If $y(0) = y_0$,

$$y = y_0 e^{-at} \tag{7.41}$$

Thus, the solution starts at y_0 and asymptotically approaches zero.

Use Euler's method

$$y_{i+1} = y_i + \frac{dy_i}{dt}h \tag{7.42}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Substituting dy/dt

$$y_{i+1} = y_i - ay_ih \tag{7.43}$$

or

$$y_{i+1} = (1 - ah)y_i \tag{7.44}$$

The stability of this formula clearly depend on the step size h. That is, $|1 - ah|$ must be less than 1.

For the fast transient part, the step size to maintain stability must be very small. In addition, an even smaller step size is required to obtain an accurate solution.

Implicit method: developed by evaluating the derivative at the future time

$$y_{i+1} = y_i + \frac{dy_{i+1}}{dt}h \tag{7.45}$$

Substituting the derivative term

$$y_{i+1} = y_i + ay_{i+1}h \tag{7.46}$$

which can be solved for

$$y_{i+1} = \frac{y_i}{1 + ah} \tag{7.47}$$

For this case, regardless of the size of the step, $|y_i| \to 0$ as $i \to \infty$. See the figure 26.2 at p 722.

7.2.2 Multistep Methods

The one-step methods utilize information at a single point x_i to predict a value of the dependent variable y_{i+1} at a future point x_{i+1}. Alternative approaches, called multistep methods, are base on information of the previous points. The curvature of the lines connecting these previous values provides information regarding the trajectory of the solution. The multistep methods exploit this information to solve ODEs.

The non-self-starting Huen method

The Heun method use Euler's method as predictor

$$y^0_{i+1} = y_i + f(x_i, y_i)h \tag{7.48}$$

and the trapezoidal rule as a corrector

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y^0_{i+1})}{2}h \tag{7.49}$$

Thus, the predictor and the corrector have local truncation errors of $O(h^2)$ and $O(h^3)$, repectively. Consequently, one way to improve Heun's method is to develop a predictor that has a local error of

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

$O(h^3)$. This can be accomplished by using Euler's method and the slope at y_i, and extra information from a previous point y_{i-1}, as in

$$y_{i+1}^0 = y_{i-1} + f(x_i, y_i)2h \tag{7.50}$$

Notice that eq. (7.50) attains $O(h^3)$ at the expense of employing a larger step size, $2h$. In addition, eq. (7.50) is not self-starting because it involves a previous value of the dependent variable y_{i-1}. Because of the fact it is called the non-self-starting Heun method.

Derivation of non-self-starting Heun method

Consider the general ODE

$$\frac{dy}{dx} = f(x, y) \tag{7.51}$$

Integrating between limits at i and $i+1$

$$\int_{y_i}^{y_{i+1}} dy = \int_{x_i}^{x_{i+1}} f(x, y)dx \tag{7.52}$$

Integrated value is

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} f(x, y)dx \tag{7.53}$$

Use trapezoidal rule to integrate the second term at right hand side

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1})}{2}h \tag{7.54}$$

which is the corrector equation for the Heun method and the trapezoidal rule gives the local truncation error of $O(h^3)$.

A similar approach can be used to derive the predictor. For this case, the integration limits are from $i-1$ to $i+1$.

$$\int_{y_{i-1}}^{y_{i+1}} dy = \int_{x_{i-1}}^{x_{i+1}} f(x, y)dx \tag{7.55}$$

which can be integrated and rearranged to yield

$$y_{i+1} = y_{i-1} + \int_{x_{i-1}}^{x_{i+1}} f(x, y)dx \tag{7.56}$$

Use the first Newton-Cote open integration formula

$$\int_{x_{i-1}}^{x_{i+1}} f(x, y)dx = 2hf(x_i, y_i) \tag{7.57}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

which is called the midpoint method.

$$y_{i+1} = y_{i-1} + 2hf(x_i, y_i) \tag{7.58}$$

which is the predictor for the non-self-starting Heun.

Integration formulas

The non-self-starting Heun method employs an open integration formula (the midpoint method) to make an initial estimate. This predictor step requires a previous data point. Then, a closed integration formula (the trapezoidal rule) is applied iteratively to improve the solution.

Newton-Cotes formulas estimate the integral over an interval spanning several points. In constrast, the Adam formulas use a set of points from an interval to estimate the integral solely for the last segment in the interval. See the figure 26.7 p 734.

- Newton-Cotes Formulas
  - Open formulas
  $$y_{i+1} = y_{i-n} + \int_{x_{i-n}}^{x_{i+1}} f_n(x)dx \tag{7.59}$$
  if $n = 1$
  $$y_{i+1} = y_{i-1} + 2hf_i \tag{7.60}$$
  which is referred to as the midpoint method and was used previously as the predictor in the non-self-starting Heun method.
  - Closed formulas
  $$y_{i+1} = y_{i-n+1} + \int_{x_{i-n+1}}^{x_{i+1}} f_n(x)dx \tag{7.61}$$
  if $n = 1$
  $$y_{i+1} = y_i + \frac{h}{2}(f_i + f_{i+1}) \tag{7.62}$$
  which is equivalent to the trapezoidal rule.
- Adams Formulas : Many popular computer algorithms for multistep solution of ODEs are based on these methods.
  - Open formulas(Adams-Bashforth) : start with a forward Taylor series expansion at x_i
  $$y_{i+1} = y_i + f_i h + \frac{f_i'}{2}h^2 + \frac{f_i''}{6}h^3 + \cdots \tag{7.63}$$
  which can also be written as
  $$y_{i+1} = y_i + h\left(f_i + \frac{h}{2}f_i' + \frac{h^2}{3!}f_i'' + \cdots\right) \tag{7.64}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Use a backward difference

$$f'_i = \frac{f_i - f_{i-1}}{h} + \frac{f''_i}{2}h + O(h^2) \tag{7.65}$$

Then

$$y_{i+1} = y_i + h\left(\frac{3}{2}f_i - \frac{1}{2}f_{i-1}\right) + \frac{5}{12}h^3 f''_i + O(h^4) \tag{7.66}$$

– Closed formulas(Adams-Moulton) : start with a backward Taylor series around x_{i+1}

$$y_i = y_{i+1} - f_{i+1}h + \frac{f'_{i+1}}{2}h^2 + \cdots \tag{7.67}$$

Solving for y_{i+1} yields

$$y_{i+1} = y_i + h\left(f_{i+1} - \frac{h}{2}f'_{i+1} + \cdot\right) \tag{7.68}$$

Use a difference to approximate the first derivative

$$f'_{i+1} = \frac{f_{i+1} - f_i}{h} + \frac{f''_{i+1}}{2}h + h(O^2) \tag{7.69}$$

Then

$$y_{i+1} = y_i + h\left(\frac{1}{2}f_{i+1} + \frac{1}{2}f_i\right) - \frac{1}{12}h^3 f''_{i+1} - O(h^4) \tag{7.70}$$

7.3 Boundary-Value and Engenvalue Problems

Boundary-value : which is specified at the extreme points or boundaries of a system.

Classification of boundary condition

- Dirichlet condition : the value of independent variable is specified at a boundary
- Neumann condition : the value of the derivative of independent variable is specified at a boundary

7.3.1 General Methods of Boundary-Value Problems

The shooting method: based on converting the boundary-value problem into an equivalent initial-value problem. A trial-and-error approach is then implemented to solve the initial-value version.

Finite-difference methods: finite divided differences are substituted for the derivatives in the original equation. Thus, a linear differential equation is transformed into a set of simultaneous algebraic equations.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

7.3.2 ODEs and Eigenvalues with Libraries and Packages

- Matlab
  - ode23
  - ode45
- IMSL
  - IVPRK
  - IVPAG
  - BVPFD
  - BVPMS

7.4 Engineering Applications: Ordinary Differential Equations

See the textbook

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Chapter 8

Partial Differential Equations

An equation involving partial derivatives of an unknown function of two or more independent variables is called a partial differential equation, PDE. The order of a PDE is that of the highest-order partial derivative appearing in the equation.

A general linear second-order differential equation is

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D = 0 \tag{8.1}$$

Depending on the values of the coefficients of the second-derivative terms eq. (8.1) can be classified int one of three categories.

- $B^2 - 4AC < 0$: Elliptic
$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{8.2}$$
Laplace equation(steady state with two spatial dimensions)

- $B^2 - 4AC = 0$: Parabolic
$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2} \tag{8.3}$$
Heat conduction equation(time variable with one spatial dimension)

- $B^2 - 4AC > 0$: Hyperbolic
$$\frac{\partial^2 y}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y}{\partial t^2} \tag{8.4}$$
Wave equation(time variable with one spatial dimension)

8.1 Finite Difference: Elliptic Equations

Elliptic equations in engineering are typically used to characterize steady-state, boundary-value problems.

8.1.1 The Laplace Equations

- The Laplace equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{8.5}$$

- The Poisson equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x, y) \tag{8.6}$$

8.1.2 Solution Techniques

- The Laplacian Difference Equation : use central difference based on the grid scheme

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} \tag{8.7}$$

  and

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta y^2} \tag{8.8}$$

  Substituting these equations into the Laplace equation gives

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta y^2} = 0 \tag{8.9}$$

  For the square grid, $\Delta x = \Delta y$, and by collecting terms

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0 \tag{8.10}$$

  This relationship, which holds for all interior point on the plate, is referred to as the Laplacian difference equation.

  This approach gives a large size of linear algebraic equations.

- The Liebmann Method : For larger-sized grids, a significant number of the terms will be zero. When applied to such sparse system, full-matrix elimination methods waste great amounts of computer memory storing these zeros. For this reason, approximate methods provide a viable approach for obtaining solutions for elliptical equation. The most commonly employed approach is Guass-Seidel, which when applied to PDEs is also referred to as Liebmann's method.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

8.1.3 Boundary Conditions

- Derivative boundary conditions : including the derivative boundary conditions into the problem.
- Irregular boundaries : use contstants to depict the curvature.

8.1.4 The Control Volume Approach

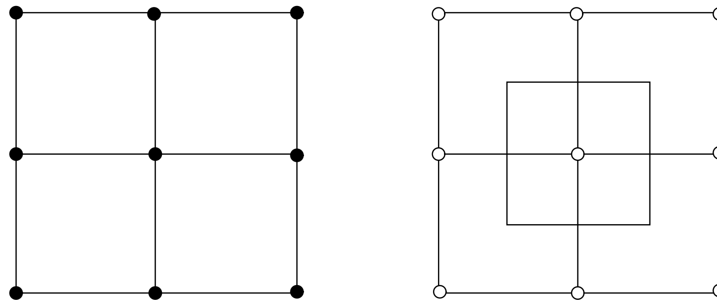

Figure 8.1: Two different perspectives for developing approximate solutions of PDEs.

Two different developing approximate solutions of PDEs

- Finite-difference : divides the continuum into nodes and convert the equations to an algebraic form.
- Control volume : approximates the PDEs with a volume surrounding the point.

8.2 Finite Difference: Parabolic Equations

8.2.1 The Heat Conduction Equation

Fourier's law of heat conduction

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2} \tag{8.11}$$

which is the heat-conduction equation.

Problems of parabolic equation

- consider changes in time as well as in space

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- temporally open-ended
- consider the stability problem

8.2.2 Explicit Methods

With finite divided differences

$$\frac{\partial T}{\partial t} = \frac{T_i^{l+1} - T_i^l}{\Delta t} \tag{8.12}$$

and

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^l - 2T_i^l + T_{i-1}^l}{(\Delta x)^2} \tag{8.13}$$

which give

$$T_i^{l+1} = T_i^l + \lambda(T_{i+1}^l - 2T_i^l + T_{i-1}^l) \tag{8.14}$$

where $\lambda = k\Delta t/(\Delta x)^2$.

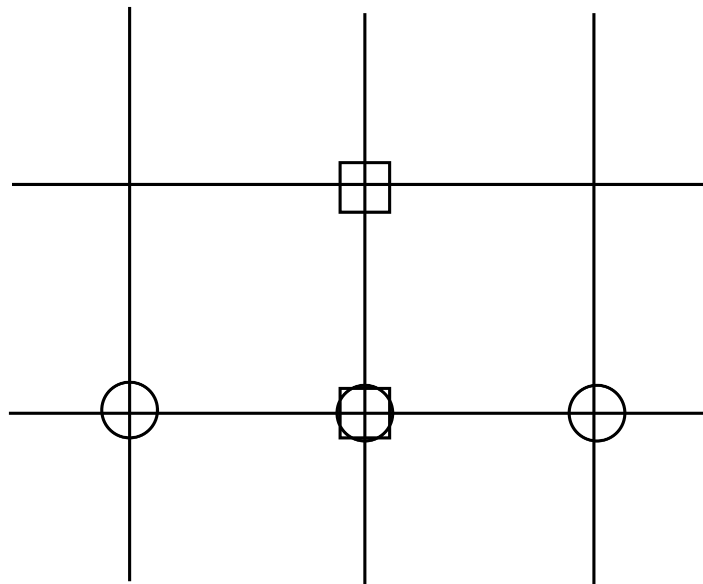

Figure 8.2: A computational modelcule for the explicit form.

Convergence and Stability

- Convergence : as Δx and Δt approach zero, the results of the finite-difference technique approach the true solution.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- Stability : errors at any stage of the computation are not amplified but are attenuated as the computation progresses.

Convergence: consider the following unsteady-state heat-flow equation in one dimension.

$$\frac{\partial U}{\partial t} = \frac{k}{c\rho}\frac{\partial^2 U}{\partial x^2} \tag{8.15}$$

Let the symbol U to represent the exact solution and u to represent the numerical solution. Let $e_i^j = U_i^j - u_i^j$, at the point $x = x_i$, $t = t_j$. By the explicit method,

$$u_i^{j+1} = r(u_{i+1}^j + u_{i-1}^j) + (1-2r)u_i^j \tag{8.16}$$

where $r = k\Delta t / c\rho(\Delta x)^2$. Substituting $u = U - e$ into the above equation,

$$e_i^{j+1} = r(e_{i+1}^j + e_{i-1}^j) + (1-2r)e_i^j - r(U_{i+1}^j + U_{i-1}^j) - (1-2r)U_i^j + U_i^{j+1} \tag{8.17}$$

By using Taylor series expansion,

$$U_{i+1}^j = U_i^j + \left(\frac{\partial U}{\partial x}\right)_{i,j}\Delta x + \frac{(\Delta x)^2}{2}\frac{\partial^2 U(\xi_1, t_j)}{\partial x^2}, \quad x_i < \xi_1 < x_{i+1} \tag{8.18}$$

$$U_{i-1}^j = U_i^j - \left(\frac{\partial U}{\partial x}\right)_{i,j}\Delta x + \frac{(\Delta x)^2}{2}\frac{\partial^2 U(\xi_2, t_j)}{\partial x^2}, \quad x_{i-1} < \xi_2 < x_i \tag{8.19}$$

$$U_i^{j+1} = U_i^j + \Delta t\frac{\partial U(x_i, \eta)}{\partial t}, \quad t_j < \eta < t_{j+1} \tag{8.20}$$

Substituting these into (8.17) and simplifying

$$e_i^{j+1} = r(e_{i+1}^j + e_{i-1}^j) + (1-2r)e_i^j + \Delta\left[\frac{\partial U(x_i, \eta)}{\partial t} - \frac{k}{c\rho}\frac{\partial^2 U(\xi, t_j)}{\partial x^2}\right],$$
$$t_j \le \eta \le t_{j+1}, x_{i-1} \le \xi \le x_{i+1} \tag{8.21}$$

Let E^j be the magnitude of the maximum error in the row of calculation for $t = t_j$, and let $M > 0$ be an upper bound for the magnitude of the expression. If $r \le \frac{1}{2}$, all the coefficients in the above equation are positive (or zero) and we may write the inequality

$$|e_i^{j+1}| \le 2rE^j + (1-2r)E^j + M\Delta t = E^j + M\Delta t \tag{8.22}$$

This is true for all the e_i^{j+1} at $t = t_{j+1}$, so

$$E^{j+1} \le E^j + M\Delta t \tag{8.23}$$

This is true at each time step,

$$E^{j+1} \le E^j + M\Delta t \le E^{j-1} + 2M\Delta t \le \cdots \le E^0 + Mt_{j+1} = Mt_{j+1} \tag{8.24}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

because E^0, the errors at $t = 0$, are zero, as U is given by the initial conditions.

As $\Delta x \to 0$, $\Delta t \to 0$ if $k\Delta t/c\rho(\Delta x)^2 \leq \frac{1}{2}$, and $M \to 0$, because, as both Δx and Δt get smaller

$$\left[\frac{\partial U(x_i,\eta)}{\partial t} - \frac{k}{c\rho}\frac{\partial^2 U(\xi,t_j)}{\partial x^2}\right] \to \left(\frac{\partial U}{\partial t} - \frac{k}{c\rho}\frac{\partial^2 U}{\partial x^2}\right)_{i,j} = 0 \tag{8.25}$$

Consequently, the explicit method is convergent for $r \leq \frac{1}{2}$, because the errors approach zero as Δt and Δx are made smaller.

8.2.3 A Simple Implicit Method

The problems of explicit finite-difference formulation

- stability
- exclusion of information that has a bearing on the solution

See figure 30.6 at p838.

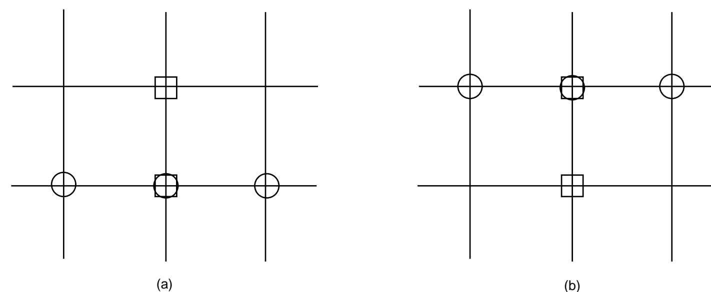

Figure 8.3: Computational molecules demonstrating the funcdamental differences.

In implicit methods, the spatial derivative is approximated at an advanced time level $l + 1$.

$$\frac{\partial^2 T}{\partial x^2} \simeq \frac{T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1}}{(\Delta x)^2} \tag{8.26}$$

When this relationship is substituted into the original PDE, the resulting difference equation contains several unknowns. Thus, it cannot be solved explicitly by simple algebraic rearrangement. Instead, the entire system of equations must be solved simultaneously. This is possible because, along with the boundary conditions, the implicit formulations result in a set of linear algebraic equations with the same number of unknowns.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

8.2.4 The Crank-Nicholson Method

The Crank-Nicolson method provides an alternative implicit scheme that is second-order accurate in both space and time. To do this, develops difference approximations at the midpoint of the time increment.

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{2}\left[\frac{T_{i+1}^{l} - 2T_i^l + T_{i-1}^l}{(\Delta x)^2} + \frac{T_{i+1}^{l+1} - 2T_i^{l+1} + T_{i-1}^{l+1}}{(\Delta x)^2}\right] \tag{8.27}$$

Substituting and collecting terms gives

$$-\lambda T_{i-1}^{l+1} + 2(1+\lambda)T_i^{l+1} - \lambda T_{i+1}^{l+1} = \lambda T_{i-1}^{l} + 2(1-\lambda)T_i^{l} + \lambda T_{i+1}^{l} \tag{8.28}$$

8.3 Finite Element Method

- Finite-difference method
  - divide the solution domain into a grid of discrete points or nodes
  - write the PDE for each node and replace the derivative with finite divided differences
  - it is hard to apply for system with irregular geometry, unusual boundary conditions, or heterogeuous composition
- Finite-element method
  - divide the solution domain into simply shaped regions or "elements".
  - develop an approximate solution for the PDE for each of these elements.
  - link together the individiual solutions

8.3.1 Calculus of variation

The calculus of variations involves problems in which the quantity to be minimized appears as an integral. As the simplest case,

$$J = \int_{x_1}^{x_2} f(y, y_x, x)dx \tag{8.29}$$

Let J is the quantity that takes on an extreme value. Under the integral sign, f is a known function of the indicated variables $y(x)$, $y_x(x)$, and x but the dependence of y on x is not fixed: that is, $y(x)$ is unknown. Thus, the calculus of variation seeks to optimize a special class of functions called functionals. A functional can be thought of as a "function of function."

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

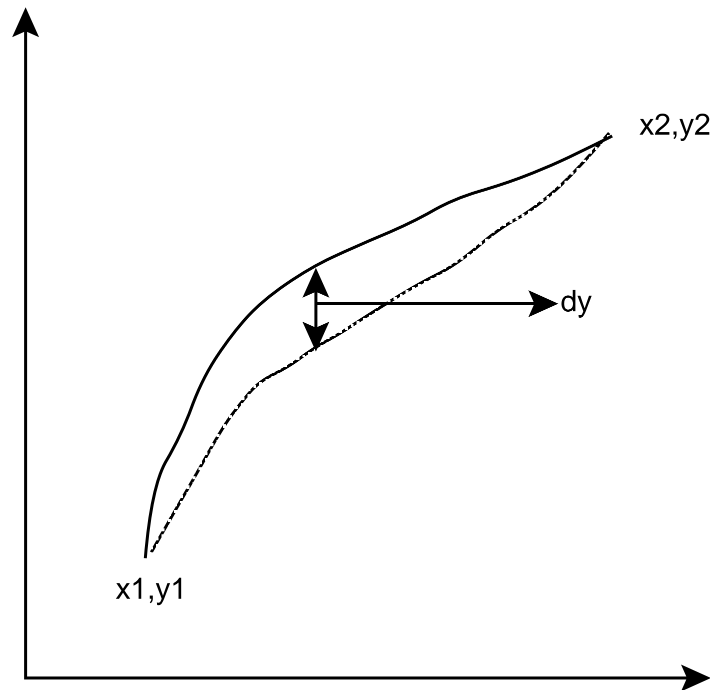

Figure 8.4: A varied path.

In figure 8.3.1 two possible paths are shown. The difference between these two for a given x is called the variation of y, δy, and introduce $\eta(x)$ to define the arbitrary deformation of the path and a scale factor α to give the magnitude of the variation. The function $\eta(x)$ is arbitrary except for two restrictions. First

$$\eta(x_1) = \eta(x_2) = 0 \tag{8.30}$$

which means that all varied paths must pass through the fixed end points. Second,

$$\begin{aligned} \eta(x) &= 1, \quad x = x_0 \\ &= 0, \quad x \neq x_0 \end{aligned} \tag{8.31}$$

With the path described with α and $\eta(x)$,

$$y(x, \alpha) = y(x, 0) + \alpha\eta(x) \tag{8.32}$$

and

$$\delta y = y(x, \alpha) - y(x, 0) = \alpha\eta(x) \tag{8.33}$$

Let $y(x, \alpha = 0)$ be the unknown path that will minimize J. Then $y(x, \alpha)$ describes a neighboring path. Then J is now a function of new parameter α:

$$J(\alpha) = \int_{x_1}^{x_2} f[y(x, \alpha), y_x(x, \alpha), x]dx \tag{8.34}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

and the extreme value is

$$\left[\frac{\partial J(\alpha)}{\partial x}\right]_{\alpha=0} = 0 \tag{8.35}$$

The partial derivative of J is

$$\frac{\partial J(\alpha)}{\partial \alpha} = \int_{x_1}^{x_2} \left[\frac{\partial f}{\partial y}\frac{\partial y}{\partial \alpha} + \frac{\partial f}{\partial y_x}\frac{\partial y_x}{\partial \alpha}\right] dx \tag{8.36}$$

From eq. (8.32)

$$\frac{\partial y(x,\alpha)}{\partial \alpha} = \eta(x) \tag{8.37}$$

$$\frac{\partial y_x(x,\alpha)}{\partial \alpha} = \frac{d\eta(x)}{dx} \tag{8.38}$$

Equation (8.36) becomes

$$\frac{\partial J(\alpha)}{\partial \alpha} = \int_{x_1}^{x_2} \left(\frac{\partial f}{\partial y}\eta(x) + \frac{\partial f}{\partial y_x}\frac{d\eta(x)}{dx}\right) dx \tag{8.39}$$

Integrating the second term by parts

$$\int_{x_1}^{x_2} \frac{d\eta(x)}{dx}\frac{\partial f}{\partial y_x} dx = \eta(x)\left.\frac{\partial f}{\partial y_x}\right|_{x_1}^{x_2} - \int_{x_1}^{x_2} \eta(x)\frac{d}{dx}\frac{\partial f}{\partial y_x} dx \tag{8.40}$$

The integrated part is zero and

$$\int_{x_1}^{x_2} \left[\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y_x}\right] \eta(x) dx = 0 \tag{8.41}$$

Multiply α

$$\alpha\left[\frac{\partial J}{\partial \alpha}\right]_{\alpha=0} = \int_{x_1}^{x_2} \left[\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y_x}\right] \delta y dx = \delta J = 0 \tag{8.42}$$

The condition for stationary is

$$\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y_x} = 0 \tag{8.43}$$

which is known as the Euler(or Euler-Lagrange) equation.

8.3.2 Example: The shortest distance between two points

We have to determine the path that minimize the distance between two points which are given as (x_1, y_1) and (x_2, y_2).

$$\int_{x_1}^{x_2} \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2} = \int_{x_1}^{x_2} \sqrt{(dx)^2 + (dy)^2} = \int_{x_1}^{x_2} \sqrt{1 + \left(\frac{dy}{dx}\right)^2} dx \tag{8.44}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Equation (8.44) is a funtional that is a function of path $y(x)$ and our problem is to find $y = y(x)$ which will make

$$J = \sqrt{1 + y'^2} dx \tag{8.45}$$

as small as possible. And $y(x)$ is called an extremal. Assuming a neighboring line

$$Y(x) = y(x) + \alpha\eta(x) \tag{8.46}$$

Select a $Y(x)$ make

$$J = \sqrt{1 + Y'^2} dx \tag{8.47}$$

a minimum. Now I is a function of the parameter α; when $\alpha = 0$, $Y = y$. Our problem then is to make $I(\alpha)$ take its minimum value when $\alpha = 0$.

$$\frac{dJ}{d\alpha} = 0 \quad \text{when} \quad \alpha = 0 \tag{8.48}$$

Differentiating gives

$$\frac{dJ}{d\alpha} = \int_{x_1}^{x_2} \frac{1}{2} \frac{1}{\sqrt{1 + Y'^2}} 2Y' \left(\frac{dY'}{d\alpha} \right) dx \tag{8.49}$$

Y' is

$$Y'(x) = y'(x) + \alpha\eta'(x) \tag{8.50}$$

Then

$$\frac{dY'}{d\alpha} = \eta'(x) \tag{8.51}$$

Put $\alpha = 0$

$$\left(\frac{dI}{d\alpha} \right)_{\alpha=0} = \int_{x_1}^{x_2} \frac{y'(x)\eta'(x)}{\sqrt{1 + y'^2}} dx = 0 \tag{8.52}$$

Integrate by part

$$\left(\frac{dJ}{d\alpha} \right)_{\alpha=0} = \frac{y'}{\sqrt{1 + y'^2}} \eta(x) \Bigg|_{x_1}^{x_2} - \int_{x_1}^{x_2} \eta(x) \frac{d}{dx} \left(\frac{y'}{\sqrt{1 = y'^2}} \right) dx \tag{8.53}$$

The first term is zero and because $\eta(x)$ is arbitarary function,

$$\frac{d}{dx} \left(\frac{y'}{\sqrt{1 + y'^2}} \right) = 0 \tag{8.54}$$

In the Euler equation case,

$$F = \sqrt{1 + y'^2} \tag{8.55}$$

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Then

$$\frac{\partial F}{\partial y'} = \frac{y'}{\sqrt{1+y'^2}}, \quad \frac{\partial F}{\partial y} = 0 \tag{8.56}$$

and the Euler equation gives

$$\frac{d}{dx}\left(\frac{y'}{\sqrt{1+y'^2}}\right) = 0 \tag{8.57}$$

From this

$$\frac{y'}{\sqrt{1+y'^2}} = c \tag{8.58}$$

Solving for y'

$$y' = \sqrt{\frac{c^2}{1-c^2}} = b \tag{8.59}$$

and

$$y = bx + a \tag{8.60}$$

8.3.3 The Rayleigh-Ritz Method

It is based on an elegant branch of methematics, the calculus of variations. With this method we solve a boundary-value problem by approximating the solution with a finite linear combination of simple basis functions that are chosen to fulfill certain ceiteria, including meeting the boundary conditions.

For example, consider the second-order linear boundary-value problem over $[a, b]$:

$$y'' + Q(x)y = F(x), \quad y(a) = y_0, \quad y(b) = y_n \tag{8.61}$$

The functional that corresponds to the above equation is

$$J[u] = \int_a^b \left[\left(\frac{du}{dx}\right)^2 - Qu^2 + 2Fu\right] dx \tag{8.62}$$

We can transform eq. (8.62) to eq. (8.61) through the Euler-Lagrange conditions, so optimizing (8.62) give the solution to eq. (8.61).

The benifits of operating with the functional rather than the original equation:

- only the first-order instead of second-order derivative
- simplify the mathematics and permits to find solutions even when there are discontinuities that cause y not to have sufficiently high derivatives.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

If we know the solution to our differential equation, substituting it for the solution will make J a minimum. Let $u(x)$, which is the approximation to $y(x)$, be a sum:

$$u(x) = c_0\nu_0(x) + c_1\nu_1(x) + \cdots + c_n\nu_n(x) = \sum_{i=0}^{n} c_i\nu_i(x) \tag{8.63}$$

Two conditions on the ν's which is called as trial function.

- chosen such that $u(x)$ meets the boundary conditions
- ν's are linearly independent.

Now fina a way of getting values fro the c's to force $u(x)$ to be close to $y(x)$ using the functional.

$$J(c_0, c_1, \ldots, c_n) = \int_a^b \left[\left(\frac{d}{dx} \sum c_i\nu_i \right)^2 - Q(\sum c_i\nu_i)^2 + 2F \sum c_i\nu_i \right] dx \tag{8.64}$$

To minimize J, take its partial derivatives with respect to each unknown c and set to zero.

$$\frac{\partial J}{\partial c_i} = \int_a^b 2\left(\frac{du}{dx}\right)\frac{\partial}{\partial c_i}\left(\frac{du}{dx}\right)dx - \int_a^b 2Qu\left(\frac{\partial u}{\partial c_i}\right)dx + 2\int_a^b F\frac{\partial u}{\partial c_i}dx \tag{8.65}$$

8.3.4 The Collocation and Galerkin Method

The collocation method is another way to approximate $y(x)$ which is called a "residual method."

$$R(x) = y'' - Qy - F \tag{8.66}$$

Algorithm of the collocation method:

- approximate $y(x)$ with $u(x)$ equal to a sum of trial function, usually chosen as linearly independent polynomials.
- substitute $u(x)$ int $R(x)$ and attempt to make $R(x) = 0$ by a suitable choice of the coefficients in $u(x)$.

Like collocation, Galerkin method is a "residual method" that use the $R(x)$, except that now we multiply $R(x)$ by weighting function, $W_i(x)$.

$$\int_a^b W_i(x)R(x)dx = 0, \quad i = 0, 1, \ldots, n \tag{8.67}$$

The advantages of collocation and Galerkin method

- amount of arithmetic is cetainly less
- much easier and never have to find the variational form.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

8.3.5 Finite elements for ordinary-differential equations

The disadvantages of the previous methods

- Find a good trial function(it it not so easy)
- polynomial may interpolate poorly.

The remedy to the above problems is based on the observation that even low-degree polynomials can reflect the behavior of a function if based on values that are closely spaced.

1. subdivide $[a, b]$ into n subintervals, called elements, that join at $x_1, x_2, \ldots, x_{n-1}$ which are called the nodes of the interval.
2. apply the Galerkin method to each element separately to interpolate between the end nodal values, $u(x_{i-1})$ and $u(x_i)$, where these u's are approximations to the $y(x_i)$'s.
3. use a low-degree polynomial for $u(x)$.
4. combine the separate element equations
5. adjust for the boundary conditions and solve equations to get approximations to $y(x)$ at the nodes.

8.4 Engineering Applications: Partial Differential Equations

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Appendix A

Using Matlab

이번 쳅터과 다음쳅터에서는 다루는 내용은 수치해석에서 사용하는 도구 가운데 Matlab과 fortran의 사용법에 대해서 배운다. Matlab의 경우에는 이미 시중에 여러가지 책이 나와있지만 대부분 제어와 관련된 내용이 많다. 이 장에서는 수칙해석에 관련해서 기본적인 사용법에 대해 배운다.

Matlab는 일반적으로 제어를 연구하는 사람들이 좋은 툴로서 알려져 있지면 현재는 공학 및 여러가지 연구하는 사람들에게 유용한 여러가지 toolbox를 포함하고 있으므로 굳이 제어만을 위한 도구라고 얘기할 수 없다. matlab은 여러가지 toolbox를 포함하고 있지만 여러분들은 기본적인 것에 대한 것만 익히는 것으로 충분하다.

A.1 설치

기본적으로 설치해야될 것들은 다음과 같다. 실질적으로 수치해석을 하는 경우에는 Simulink나 Control System Toolbox 같은 것은 필요 없을 수 있지만 기본적이 matlab의 기능을 맛보기 위해서는 필요한 것들이다. 이글을 작성하는 시기에 나온 matlab의 최신 버전은 6.1이다. 그러나 여러분들은 4.대의 버전을 사용해도 충분히 활용할 수 있다. 설치하는 방법은 다른 윈도우 프로그램을 설치하는 것과 동일한 방법으로 설치를 하면 된다. 설치한 내용을 선택한 다음 다음버튼을 눌러서 설치를 하면 된다.

- Matlab
- Ghostscript Printer Driver
- Simulink
- Control System Toolbox

- Optimization Toolbox
- Spline Toolbox
- Statistics Toolbox

A.2 Matlab 기초

Matlab은 사용하는데 있어서 상당히 편하고 느슨한 사용법을 요구한다. 변수의 선언이나 그러한 과정은 전혀 없으며 대부분의 경우 문제를 해결하기 위한 적절한 함수를 찾아 사용하면 된다. 기본으로 사용하는 명령어는 가운데 도움말을 얻을 수 있는 명령어는 `help`, `lookfor` 등이다. 이 명령어들은 사용하고자 하는 함수에 대한 사용법을 보여주고 만일 사용자가 정확한 함수명을 모르지만 원하는 기능에 대한 함수를 찾는 경우 그 함수를 찾아주는 기능이 있다.

자 이제부터 기본적인 Matlab에 대해 배워보자. Matlab을 배울때도 다른 프로그래밍 언어와 같이 제어문과 함수 작성법에 대해서만 배운다면 기본적인 내용은 다 배운셈이다. 이것외에 simulink나 여러가지 toolbox에 대해 배워야 하지만 그 부분은 대부분 초보적 단계에서나 아니면 제어를 연구하는 사람들 외에는 필요가 없는 부분이다. 현재에는 Matlab에서 여러가지 toolbox들이 나왔다. 처음에는 공학 쪽이였지만 현재는 경영학과 쪽의 toolbox도 나온 것으로 알고 있다. 이러한 toolbox들은 여러분의 필요에 따라 익히면 된다.

```
>>a=1

a =

    1
```

위의 예제는 a라는 변수에 1이라는 값을 치환한 것이다. 이러한 경우 a 변수를 정수형 변수가 된다.

```
>>a=1

a =
    1
>>a=1.1

a =

    1.1000
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

좀전에 Matlab에서는 변수에 대한 선언이 필요 없다고 했었다. 처음에는 a는 정수형 변수이고 두번째는 a를 실수형으로 사용한 것이다. 기본적으로 숫자에 대한 포멧 형태가 short이기 때문에 실수형으로 선언한 경우에 1.1000이 되었다. 만약 이렇게 숫자의 포멧 형태를 바꾸고 싶은 경우에는 `format long`이라는 명령어로 수정을 하면된다. 물론 이경우에는 실수형 변수를 의미한다.

```
>>format long
>>a=1.1

a =

   1.10000000000000
```

지수형 형태의 결과물을 얻고 싶은 경우에는 `format short(long) e`라고 하면 된다. 이해가 가지 않거나 사용법을 잘 모르는 경우에는 모두 "`help 명령어`"를 하면된다.

Matlab에서는 사용자가 입력한 결과를 계속해서 되풀이해서 보여준다.(echoing 한다는 것이다.) 만일 명령 수행 결과를 보고 싶지 않은 경우에는 명령을 입력하고 semicolon ;를 붙여주면 된다. 또한 줄이 너무 길어지는 경우에는 ... 을 줄 마지막에 붙여주면 두줄이 연결되었다고 인식한다. 또한 현재 변수의 값을 알고 싶은 경우에는 단순히 변수명을 입력하고 엔터를 치면 값을 알 수 있다.

```
>>a=1;
>>b=a+10*20 ...
-20;
>>a

a =
     1
```

Matlab에서 사용하는 수식의 대부분은 우리가 이미 알고 있는 수식의 표현과 같다. 사칙연산의 경우는 `+, -, *, /`이고 대입연산자는 `=`이다. 거듭제곱의 경우는 ^ 기호를 사용한다.

마지막으로 현재 내가 사용하고 있는 변수명을 알고 싶은 경우에는 who 명령어를 사용하면 되고 변수에 대한 자세한 내용을 알고 싶은 경우에는 whos라는 명령어를 사용하면 된다. Matlab을 실행을 한 후 내가 입력하는 명령어대한 기록을 얻고 싶은 경우에는 diary라는 명령어를 사용하면 여러분이 현재 입력한 명령어와 결과값에 대한 히스토리가 모두 정해진 파일에 기록이 된다. 자세한 것은 `help diary`를 참조하기 바란다.

이렇게 텍스트 파일로 여러분이 수행한 명령어에 대한 기록 외에 여러분들이 계산한 결과 값을 저장할 때는 어떻게 해야할까 ? 즉, 위의 예제처럼 여러분은 a라는 변수와 b라는 변

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

수를 사용하였다. 이 두변수를 저장하기 위해서는 save라는 명령어를 사용한다. 이 명령어를 이용하면 원하는 변수를 저장할 수 있다. 이 경우 저장되는 파일의 확장자는 .mat이며 파일의 형태는 바이너리이다. 이러한 경우에는 파일의 내용을 일반 에디터에서는 볼 수 없다. save 명령어에 대한 도움말을 보면 변수의 값을 아스키 값으로도 저장할 수 있다. 자세한 내용은 직접 찾아보기 바란다. 저장되어있는 변수 값을 다시 matlab의 workspace로 불러오기 위해서는 load 명령어를 이용한다. 이 명령어는 mat 파일 뿐만 아니라 일반 아스키 파일도 불러올 수 있다.

```
>>a=1

a =

     1

>>b=2

b =

     2

>>c=3

c =

     3

>>save test

>>save ab a b

>>clear

>>load test

>>who

Your variables are:

a          b          c
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

위의 예제에서는 a와 b, c 라는 변수에 각각 1,2,3을 대입하고 난 후 일단 test라는 이름으로 workspace의 값들을 저장했다. `save test`라는 명령어에 의해 `test.mat` 파일이 생성이 되고 이 파일에는 a, b, c 변수의 값들이 들어 있다. 즉, save 다음에 파일명을 정해주고 변수명을 정해주지 않으면 현재의 모든 변수값을 저장하게 된다. 두번째 `save ab a b`문에 의해서는 `ab.mat` 파일에 a와 b의 값이 저장이 된다. 그러므로 원하는 변수만을 저장하는 경우에는 변수명을 정해줘야된다. 그 다음 `clear`라는 명령어를 사용했는데 이 명령어는 workspace 내의 모든 변수에 대한 정보를 지우는 명령어이다. 이 명령어에 의해 더이상의 변수는 존재하지 않는다. 그 다음 다시 `load test`라고 하면 `test.mat` 파일을 읽어서 저장된 변수명과 값을 가지고 온다.

키보드를 이용해서 이전에 사용한 명령어를 가지고 올 수 있다. 이 기능은 대부분의 리눅스 머신이나 도스의 경우 `doskey.exe` 명령어를 실행한 후 사용하는 히스토리 기능을 의미한다. 이 기능은 방향키를 이용해서 실행할 수 있다. 위 아래로 움직이는 방향키를 입력해 보면 이전에 사용했던 명령어를 볼 수 있다. 또한 ESC키를 입력하면 입력되어 있는 명령어가 지워진다. 이러한 키보드을 이용한 명령어 호출을 이용하면 복잡한 명령어를 한번만 실행하고 나면 계속적으로 이용할 수 있다.

> 사실 좀 재미있는 사실은 여러분들은 잘 모르고 있겠지만 `matlab`는 윈도우에서 사용할 수 있는 버전과 유닉스에서 사용할 수 있는 버전으로 나눠진다. 그렇기 때문에 각각의 운영체제에서 사용할 수 있는 명령어를 동시에 사용할 수 있다. 만약 현재 디렉토리에 어떤 파일이 있는가를 보기 위해서는 `dir`이라는 명령어를 도스에서는 사용하는데 유닉스나 리눅스에서는 `ls`라는 명령어를 사용한다. 그런데 이 두 명령어가 윈도우용에서도 같이 사용할 수 있다. 게다가 `pwd`라는 유닉스 명령어가 있는데 이 명령어는 현재 자신이 있는 디렉토리의 위치를 알기 위해서 사용한다. 그런데 이 명령어는 도스에는 없지만 그래도 윈도우용 `matlab`에서 사용할 수 있다.

그러면 이제 이런 경우를 생각해 보자. 여러분이 일련의 명령어들을 한번에 순차적으로 수행을 할려면 어떻게 해야할까 ? 계속해서 명령 프롬프트에 계속 입력을 해야할까 ? 그렇게 하고 나면 나중에 또 다시 사용할려면 다시 입력을 해야할까 ? 너무 지겹고 지루한 일이 되지 않을까 ? 이 부분은 여러분들이 잘 기억을 해야될 사항이다. 지금까지 배운 것을 이용해서 만들어 볼 수도 있고 계속적으로 사용할 부분이다.

```
>>a=1;
>>b=2;
>>a+b

ans =

     3
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

위와 같은 예제를 이용해서 파일을 한번 만들어보자. 여러분들이 잘 사용하는 텍스트 에디터를 이용해서 다음과 같은 sample.m이라는 파일을 다음과 같이 만들어라. 이 파일은 c:\temp\sample.m 이라고 저장이 된다고 하자. 그리고 텍스트 에디터라고 하는 것은 "MS 워드"나 아니면 "한글"을 얘기하는 것이 아니고 메모장과 같은 것을 얘기하는 것이다.

```
clear all
a=1;
b=2;
a+b
```

위와 같은 sample.m 파일을 만든 다음 matlab을 실행시켜 cd 명령어를 이용해서 c:\temp로 디렉토리를 옮겨라. 그 다음 좀전에 만든 sample.m을 실행하기 위해서는 단순히 sample이라고 명령어를 입력하면 sample.m 파일으로 로딩해서 그 파일에 있는 matlab 코드를 순차적으로 실행을 한다.

```
>>cd \temp
>>sample

ans =

     3
```

A.2.1 배열

matlab에서 벡터나 행렬을 만드는 방법은 아주 간단하다. 그리고 이 프로그램의 이름이 matlab인 것은 MATrix LABoratory라는 이름에서 왔듯이 상당히 배열과 관련된 것은 아주 쉽게 이용할 수 있다. 다음의 예제를 보자.

```
i=1:1:10;
j=0:2:12;

k=1:10;

a=[1 2];
b=[1 2;3 4];
c=[1 2 3;4 5 6];
```

처음에 나온 i라는 변수는 1부터 10까지 1크기로 증가를 하는 벡터이다. i벡터를 만들기 위해 처음에 사용된 1은 시작점을 의미하며 두번째 1은 증가분, 세번째 10은 종료점을 의미한다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

그러므로 i=$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$이며 만약 i 벡터의 3번째 값을 사용하고 싶은 경우에는 i(3)이라고 하면 된다. 그러면 만약 변수이름 없이 단순히 1:10이라고 하면 어떻게 될까 ? 이런경우에는 이름이 정해지지 않은 1부터 10까지 1씩 증가한 벡터가 얻어진다. 이 이름이 정해지지 않는 벡터에 대한 얘기를 하는 이유를 생각해 보자. 여러분이 만약 좀전에 얘기한 i라는 벡터의 두번째 요소부터 5번째 요소까지의 값을 사용할 필요가 있는 경우 어떻게 나타내야할까 ? 이런 경우에는 좀전에 얘기한 이름이 정해지지 않고 단순한 숫자의 나열을 이용하면 된다. 즉, i(2:5)라고 표기를 하면 $\begin{bmatrix} i(2) & i(3) & i(4) & i(5) \end{bmatrix}$의 부분 벡터를 만들 수 있다. 지금까지 배운것을 이용하면 j라는 벡터는 0부터 시작하여 12까지 2씩 증가하면서 만든 벡터이다. 그리고 마지막으로 증가분을 생략하는 경우에는 기본적으로 1씩 증가를 했다고 본다. 그러므로 k라는 벡터는 i 벡터와 동일한 경우가 된다.

그 다음 행렬을 만드는 방법에 대해 생각을 해보자. 사실 처음에 있는 변수 a는 행렬이 아니라 벡터이다. 기존의 ':'을 찍어서 만드는 방식은 자동으로 증가가 되는 형태이나, '[', ']' 를 이용한 벡터는 사용자가 원하는 행태로 벡터를 만들 수 있다. 각 요소간의 구분은 space로 한다. 그러므로 변수 a는 1행 2열 벡터가 된다. 행을 구분하기 위해서는 ';'이 사용된다. 변수 b는 2행 2열 행렬이 된다. 또한 c는 2행 3열 행렬이다.

이제 이러한 배열을 이용해서 간단한 계산을 해보자. 일단 지금 사용할 벡터와 행렬을 만들자.

```
>>a=1:5;
>>b=pascal(3);
>>c=magic(3);
```

a라는 벡터는 이미 알고 있겠지만 1부터 5까지 순차적으로 증가된 형태의 벡터이고 b와 c는 각각 matlab에 있는 명령어를 이용해서 만든 행렬이다. 행렬이나 벡터인 경우에 각각의 덧셈과 뺄셈은 동일하다.

```
>>b+c
>>b-c
```

위의 명령어를 입력하면 각각 두 행렬의 합과 차를 계산해 준다. 이 부분은 벡터를 이용하는 경우에도 동일하다. 그럼 행렬가운데 일부분을 사용하는 경우를 고려해보자. b 행렬의 1열과 c 행렬의 1열의 합을 계산하는 경우는 어떻게 해야될까 ?

```
>>b(:,1)+c(:,1)
```

이렇게 ' : ' 기호를 이용하면 행렬의 한 부분 모두를 의미해서 사용할 수 있다. 즉, `b(:,1)`라고 입력을 하면 b라는 행렬의 첫번째 열 모두를 의미하게 된다. 그렇다면 이것을 이용해서 행렬의 일부분은 열이나 행의 일부분을 사용할려면 어떻게 해야될까 ?

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
>>b(1:2,1:2)+c(1:2,1:2)
```

이렇게 하면 행렬의 일부분만을 이용해서 계산에 사용할 수 있다. 이렇게 사용하다가 보면 정말 다른 것은 몰라도 행렬과 관련된 계산에서는 matlab을 따라 올것이 없을 것이다.

행렬의 거듭제곱이나 나눗셈(역행렬)에 관련된 얘기를 해보자. 행렬의 곱을 계산하기 위해서는 다음과 같이 계산을 하면된다.

```
>>b*c
```

행렬의 곱에 대해서는 이미 알고 있겠지만 b 행렬과 c 행렬의 각각의 열과 행이 같은 크기를 가지고 있어야만 된다. 그부분은 matlab이 알아서 계산을 해 준다. 그 다음 행렬의 거듭제곱은 다음과 같이 계산을 하면 된다.

```
>>b*b
>>b^2
>>b^5
```

첫번째 계산은 그냥 두개의 곱을 계산을 한 것이고 두번째 것은 정말 제곱을 계산한 것이다. 그러므로 행렬의 거급제곱이나 아니면 더 높은 승을 계산하는 것은 간단히 세번째 예처럼 계산을 하면 된다. 그러면 이번에는 나눗셈을 해 보자. 이 부분은 이미 설명을 했었다. 행렬의 나눗셈에 대해서 한번 알아보자.

$$
\begin{aligned}
Ax &= B \\
x &= \frac{B}{A} \quad \leftarrow (\times) \\
x &= BA^{-1} \quad \leftarrow (\times) \\
x &= A^{-1}B \quad \leftarrow (\bigcirc)
\end{aligned}
\tag{A.1}
$$

숫자값을 이용한 계산이 아니고 행렬을 이용해서 계산하는 경우에는 위의 수식을 잘 기억해야 된다. 또한 역행렬을 앞에 붙이냐 아니면 뒤에 붙이냐고 계산 결과가 달라진다. 꼭 명심을 해야된다. 자 그럼 이제 예를 보자.

```
>>b/c
>>b\c
>>b*c^-1
>>b*inv(c)
```

첫번째 예는 그냥 생각을 해 볼 수 있는 부분이다. a라는 값을 b라는 값으로 나누라는 것처럼 보이지만 이 부분은 b라는 값으로 나눈 것이 아니라 b의 역행렬을 곱하라는 얘기이다. 그리

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

고 두번째는 b라는 값에 a의 역행렬을 곱하라는 얘기이다. 두개는 사용한 기호가 틀리므로 잘 보기 바란다. 그리고 세번째와 네번째는 첫번째와 동일한 결과를 얻는다. 일단 b라는 행렬에 -1 승을 계산한 다음 곱하는 경우이고 그리고 네번째는 b라는 행렬의 역행렬을 계산하는 inv라는 함수를 이용해서 계산한 것이다.

자 이제 행렬과 관련된 부분에 대한 설명을 거의 끝을 냈다. 마지막으로 이것 하나만 더 보자. 좀전에 행렬의 거듭제곱을 계산 했었는데 거듭제곱 말고 행렬 각각의 요소의 제곱을 계산할 때는 어떻게 해야 될까 ? 아니면 벡터가 있을 때 그 벡터의 각 요소의 제곱을 게산할려면 어떻게 해야 될까 ? 이때에는 ' . ' 기호를 이용해서 계산을 하면 된다.

```
>>a.^2
```

이 명령어는 a라는 벡터의 각 요소의 값의 제곱을 구하라는 명령어이다. 결과로는 $\begin{bmatrix} 1 & 4 & 9 & 16 & 25 \end{bmatrix}$가 얻어진다. 또 다른 예제로서 벡터의 각 요소의 2배의 값을 계산해야되는 경우는 어떻게 해야 될까 ?

```
>>2*a
```

이렇게 하면 된다. 그렇지만 만약 다음과 같은 경우를 생각해 보자.

```
>>aa=1:2:9;
>>aa * a;
>>aa .* a;
```

행렬이나 벡터든 곱을 계산하기 위해서는 그 크기를 계산해서 다른 경우에는 계산이 되지 않는다. 그러므로 첫번째 계산은 시행이 되지 않는다. 그렇지만 ' . '를 사용하는 경우에는 얘기가 틀려진다. 이것을 이용하는 경우에는 행렬이나 벡터에서 크기를 따지지 않고 단지 그 크기가 같은가만을 따진다. 즉, 계산할 수 있느나 없느냐만을 고려하기 때문에 두번째 계산은 시행이 된다. 계산 결과는 각각의 요소의 곱을 계산해 준다. 이 부분은 나눗셈을 계산할때도 틀려진다. 왜냐면 행렬의 나눗셈은 역행열을 곱하는 것인데 만약 ' ./ '를 사용한다면 각 요소의 나눗셈을 계산하는 결과를 얻을 수 있다. 이러한 계산 방법이 유용할 때가 있을 것이다.

A.2.2 Customization

Matlab을 사용하다가 보면 몇가지 마음에 안드는 부분이 있을꺼다. 일단 이러한 경우를 생각해 보자. 사실 여러분은 autoexec.bat와 config.sys 두 파일에 대해서 잘 모를것이다. 예전에 도스를 사용하던 때에는 아주 중요한 파일이 이 두가지이다. 왜 중요하냐면 컴퓨터를 사용하는 기본적인 환경을 만들어주는 파일이기 때문이다. 그런데 지금과 같은 컴퓨터 사용환경에서는 거의 사용되지 않지만 그래도 포트란 같은 언어를 사용하는 경우 몇가지 설정해줘야하는

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

사항이 있기는 하다. 그런데 매트랩에서도 이러한 초기 설정 파일이 필요하지 않을까 ? 만약 있다면 어디에 어떤 파일을 수정해야만 나만의 설정 파일을 만들 수 있을까 ? 궁금하면 일단 여러분의 매트랩 버전을 체크해보자. 필자의 경우에는 5.3을 사용하고 있기때문에 기본적인 설치 디렉토리는 `C:\MATLABR11`이다. 여러분의 상황에 따라 조금씩 다르겠지만 아마도 적당한 위치에 설치되어 있을 것이다. 그러면 일단 매트랩이 설치된 디렉토리에 가서 파일 찾기로 `matlabrc.m`이라는 파일을 찾아보기 바란다. 이 파일이 설정파일이다. 필자의 경우에는 `C:\MATLABR11\toolbox\local`이라는 디렉토리에 다음과 같은 내용으로 되어 있다.

```
%MATLABRC Master startup M-file.
%   MATLABRC is automatically executed by MATLAB during startup.
%   It establishes the MATLAB path, sets the default figure size,
%   and sets a few uicontrol defaults.
%
%       On multi-user or networked systems, the system manager can put
%       any messages, definitions, etc. that apply to all users here.
%
%   MATLABRC also invokes a STARTUP command if the file 'startup.m'
%   exists on the MATLAB path.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 1.94 $  $Date: 1998/08/24 19:53:59 $

% Set up path.
if exist('pathdef','file')
  matlabpath(pathdef);
end
.
.
.
```

여러분이 수정해야될 부분은 다음과 같다.

```
%% For European countries using A4 paper the following line should
%% be uncommented
%set(0,'DefaultFigurePaperType','a4')
```

이 부분은 일반적으로 미국에서는 A4용지를 사용하지 않고 Letter 크기의 용지를 사용한다. 그러나 여러분들은 한국에 있고 또한 A4용지를 사용하므로 위의 부분을 다음과 같이 수정한다.

```
%% For European countries using A4 paper the following line should
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
%% be uncommented
set(0,'DefaultFigurePaperType','a4')
```

이미 얘기했지만 매트랩에서 주석은 '%'으로 표시하면 된다. 내용이 바뀐부분은 주석으로 처리된 부분을 활성화 시킨것이다. 그 다음 수정할 부분은 사실 선택사항인데 만약 기본적인 형태로 매트랩을 사용하는 경우에는 매트랩을 실행하고 난 후 pwd라는 명령어를 이용해서 보면 다음과 같다.

```
  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>pwd

ans =

C:\MATLABR11\work

>>
```

이런식으로 보일꺼다. 즉, 현재 디렉토리는 `C:\MATLABR11\work`에 있다. 그런데 이 부분을 수정하고 싶으면 matlabrc.m 파일의 맨 마지막부분에 다음과 같은 말을 추가하면 된다. 일단 수정전에는 다음과 같다.

```
% Execute startup M-file, if it exists.
if exist('startup','file')
    startup
end
```

이 마지막 부분은 사용자가 만든 startup 파일 있으면 그것을 실행하라는 얘기이다. 특별히 더 수정해야될 부분이 있으면 하면 된다. 이 부분의 뒤에 다음과 같은 부분을 추가해라.

```
% Execute startup M-file, if it exists.
if exist('startup','file')
    startup
end

cd \tclee\matlab\lecture
```

맨 마지막 부분은 모든 기본적인 것이 다 실행이 된 후 디렉토리를 적당한 곳으로 옮기라는 것이다. 필자의 경우에는 `C:\tclee\matlab\lecture` 라는 디렉토리에 내가 필요로 하

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

는 모든 것을 저장해 두고 또한 작업도 이 디렉토리에서 하기 때문에 그 디렉토리로 이동을 했다. 이렇게 해 두면 원하는 디렉토리에 기본적인 파일을 다 저장하고 사용할 수 있다.

마지막으로 path에 대한 것을 생각해보자.

A.2.3 Summary

이번장에 배운 내용을 요약하면 다음과 같다.

- help와 lookfor 명령어
- 변수의 형태는 주어진 값에 의해 결정된다.
- 변수 값의 저장 방법 및 불러들이기, 제거 방법
- format 명령어의 사용법에 대해 익힌다.
- ;와 ... 의 사용법에 대해 익힌다.
- who와 whos 명령어
- diary 명령어
- 파일을 작성하고 부르는 방법
- Matlab 설정 변경 방법

A.3 제어문

전장에서 Matlab을 어떻게 사용하는지 아주 기본적인 것에 대해서는 배웠다. Matlab은 사실 그렇게 배워야 되는 것이 많은 언어는 아니므로 몇가지만 더 해보기로 한다.

언어를 배우는데 있어서 기본적으로 필요한 것들은 데이터의 선언방법과 제어문 작성법, 서브루틴 작성법이다. 이 세가지 정도만 안다면 일단 시작을 할 수 있다. 바로 전 장에서 데이터의 선언은 기본적인 형태가 없고 그냥 선언에 따라 변한다고 했다. 그렇다면 이번에는 제어문 작성 방법에 대해 배워보자.

A.3.1 if, else, and elseif

if문은 가장 기본적인 제어문의 형태이다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
if logical_expression
   statements
end
```

여기서 logical_expression에서는 if문이 실행될 조건이 들어있다. 어떤 조건일때 밑에 있는 statements를 실행할지에 대해 조건을 줘야만 한다. 예제를 보자.

```
if a==3
   b=0.2;
end
```

위의 예제는 단순하게 a라는 변수의 값이 3이 되는 경우에 b라는 변수에 0.2라는 값을 대입을 하는 예제이다. 이 예제를 한줄에 쓸 수도 있는데

```
if a==3, b=0.2; end
```

comma(,) 기호나 ouput suppression(;) 기호를 사용하는 경우에는 한줄에 여러가지 명령어를 사용할 수 있다. else까지 포함하는 경우의 기본 형태는 다음과 같다.

```
if logical_expression
   statements 1
else
   statements 2
end
```

이 경우는 logical_expression에 맞는 경우에는 1번 문장이 수행되고 아닌 경우에는 2번 문장이 수행이 된다. 마지막으로 elseif를 사용하는 경우에 대해 설명하겠다. 이 경우는 나중에 나올 switch 문과 유사하다.

```
if logical_expression 1
   statements 1
elseif logical_expression 2
   statements 2
elseif logical_expression 3
   statements 3
end
```

좀전에 배운 else 경우에는 logical_expression이 들어가지 않았지만 elseif의 경우에는 여러개의 logical_expression이 들어갈 수 있다. 다음 예제를 보자

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
if n < 0
   disp('Negative');
elseif n == 0
   disp('Zero');
else
   disp('Positive');
end
```

이 예제에서 disp라는 명령어를 사용했는데 이 함수는 화면에 원하는 문장을 출력하는 함수이다. n이라는 변수의 값에 따라 음수인지 아니면 0이거나 양수인지를 화면에 뿌려주는 예제이다. 지금까지의 내용은 if문의 사용방법에 대해서 설명을 했다. 다음에는 switch문에 대해 설명을 해 보자.

A.3.2 switch

switch문은 여러개의 if-elseif를 사용하는 경우와 매우 비슷하다. 기본적인 형태는 다음과 같다.

```
switch expression
   case value1
      statements 1
   case value2
      statements 2
   .
   .
   .
   otherwise
      statements n
end
```

위의 기본 형태에서 expression의 값이 각각의 value와 일치하는 것에 따라 statement 를 실행을 한다. 그리고 기본적으로는 otherwise 다음의 명령어들을 실행하다. if-elseif를 사용하는 경우보다 깔끔하게 프로그램을 작성할 수 있다. 예제를 보자.

```
switch a
   case 1
      disp('a is 1');
   case -1
      disp('a is -1');
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
    case 0
       disp('a is 0');
    otherwise
       disp('something else !');
end
```

a의 값에 따라 다른 문장을 실행시켜준다.

A.3.3 while

while문은 좀 뒤에 설명이 될 for와 유사한 제어문이다. 조건에 맞을때까지 계속적으로 순차적으로 명령문들을 수행한다. 기본적인 형태를 한번 보자.

```
while expression
   statement
end
```

위에서 조건은 expression에 넣고 수행되어야할 문장은 statement에 넣어서 사용한다. 예를 들어 1부터 100 까지의 합을 계산하는 경우를 해 보자.

```
n=0;
sum=0;
while n < 100
   n=n+1;
   sum=sum+n;
end
disp([n sum]);
```

위의 예제는 처음에 n과 sum 변수를 각각 0으로 하고 while 제어문에 의해 n을 증가시켜가면서 sum이라는 변수에 값을 더해간다. 이러한 반복적인 제어문을 사용할때는 조건을 잘 살펴봐야한다. 잘못하는 경우에는 원하는 결과를 얻을 수 없는 경우가 있다. 예를 들어 n=1이라고 하는 경우를 생각해보자.

```
n=1;
sum=0;
while n < 100
   n=n+1;
   sum=sum+n;
end
disp([n sum]);
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

위의 예제는 좀전에 얘기한 예제와 아주 유사해 보인다. 그렇지만 결과값은 틀리게 나온다. 하나씩 분석을 해보자. n=1이라고 했기때문에 while문에 들어오면 n은 처음에 2라는 값을 가지게 된다. 그러면 sum에 더해지는 값은 n이 1인 값이 더해지는 것이 아니라 2는 값부터 더해지게 된다. 그렇다면 원하는 결과를 얻을 수 없을 것이다. 그러면 다시 한번 n=0이라고 하고 while문에서 순서가 바뀌는 경우를 보자

```
n=0;
sum=0;
while n < 100
   sum=sum+n;
   n=n+1;
end
disp([n sum]);
```

이 경우는 sum에 처음 더해지는 값이 0이 더해지고 그 다음 값이 진행되므로 결과는 비슷하게 나올꺼라 예상이 된다. 그렇지만 좀더 생각을 해보면 n이 99인 값을 가지고 while이 진행이 되는 경우 100보다는 작기 때문에 while문은 참이 되서 진행이 된다. 그러면 sum에 99를 더하고 n이 하나 더 진행이 되므로 100인 값을 가지고 while문의 조건을 비교한다. 그런경우 100보다 작지 않기 때문에 while문을 더이상 진행하지 않고 나온다. 즉, n이 100인 경우의 값을 더하지 않게 된다. 결과는 1부터 99까지의 값을 더한 결과가 얻어진다. 그러므로 while문이나 for문을 사용하는 경우에는 조건에 대해 잘 생각을 해 봐야된다.

A.3.4 for

사실 위에서 얘기한 while문 보다는 for문을 더 많이 사용하는거 같다. 그 이유는 for문이 좀더 직관적으로 반복제어를 할 수 있기때문이다. 기본적인 형태는 다음과 같다.

```
for index=start:increment:end
   statements
end
```

for문이 실행되기 위해서는 실행되는 횟수에 대한 index를 만들어주는데 간격에 대한 정보를 넣어줘야된다. 이 부분은 이미 배열 부분에서 설명을 했다. 어떤 변수를 배열로서 사용하기 위해서는 `var_name=start:increment:end` 형식으로 만들어야 된다. for문에서도 동일하게 인덱스를 만들어 사용하면 된다. 이제 예를 들어보자. 1부터 10까지의 숫자들에 각각의 자승을 구하는 프로그램을 구성해 보자.

```
for i=1:10
   j=i^2;
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
        disp([i j]);
    end
```

위의 프로그램을 실행하면

```
    >>sample
         1     1

         2     4

         3     9
    ....
```

위와 같은 결과를 얻을 것이다. 보다시피 while문 보다 좀더 직관적인 표현으로 반복 실행문을 얻을 수 있다.

A.3.5 break

수행중인 제어문을 빠져나기기 위해서는 break를 사용한다. 예를 들어 1부터 10까지의 숫자에 대한 각각의 자승을 계산하다가 만약 7이라는 값이 나오는 경우 반복 실행문을 빠져나가는 예를 만들어 보자.

```
    for i=1:10
       j=i^2;
       if(i==7), break, end
    end
```

지금까지 배운것을 바탕으로 보면 위의 프로그램은 정확히 i 값이 7이 되었을때 반복 실행을 중단하고 빠져나간다. 위와 같이 반복 실행문을 빠져나갈 필요가 있는 경우에는 break 명령을 사용한다.

A.3.6 Summary

지금까지 여러가지 제어문의 형태에 대해 배웠다.

- if, else, and elseif
- switch

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- while
- for
- break

A.4 함수만들기

좀전에도 설명을 했지만 언어를 배우는데 지금까지 배운 것들은 변수와 제어문에 대해 배웠다. 마지막으로 이제 부프로그램을 생성하고 이용하는 방법에 대해 배워보자.

함수는 function이라는 키워드와 출력변수, 입력변수, 함수이름으로 구성이 되어 있다. 기본구조를 보자.

```
function output_name = function_name(input_name)
```

output_name은 함수가 넘길 변수의 이름이고 function_name은 현재 사용하고자 하는 함수의 이름이며 input_name은 입력으로 들어올 변수의 이름이다. 예제를 들어보자. 다음 예제는 주어진 벡터의 평균을 계산하는 함수이다.

```
function mean_value=average(x)
[m,n]=size(x);
if(~((m==1) | (n==1)) | (m==1 & n==1))
   error('Input must be a vector')
end

mean_value=sum(x)/length(x);
```

이렇게 작성한 함수를 mean_value.m이라고 저장을 한 후 다음의 예제를 한번 해보자.

```
>>z=1:99;
>>mean_value(z)

ans =

    50
```

자 그러면 기본구조와 비교를 하면서 mean_value라는 함수를 하나씩 보자. 함수의 출력값은 mean_value라는 변수값으로 넘어가게 된다. 그러므로 mean_value의 값은 가장 나중에 정해지

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

는 것이 일반적일 것이다. 함수의 이름은 average라고 정했다. 여기서 좀 주의를 해서 봐야할 것이 있다. 함수의 이름은 average라고 선언을 했고 이렇게 만든 함수는 mean_value.m이라고 저장을 했다. 그 다음 함수를 이용할때는 average를 사용한 것이 아니라 mean_value라고 사용을 했다. 즉, 함수의 이름과는 별개로 저장되어 있는 경우 파일의 이름을 함수의 이름으로 사용하게 된다. 이번 예제에서는 문제를 일으키지 않았지만 좀더 복잡한 프로그램을 만들다 보면 이렇게 파일의 이름과 함수의 이름이 틀린 경우 예상치 못한 결과를 얻을 수 있으로 함수의 이름과 파일의 이름을 동일하게 만들고 사용해야된다. 명심하기 바란다. 그러므로 만약 함수의 이름이 average라면 이 파일 역시 average.m이라고 저장을 해야된다.

그리고 matlab에 환경변수 가운데 path라는 것이 있는데 이 변수에는 사용자가 입력을 한 함수명을 찾는 경로를 지정해 주고 있다. 이런데 이 변수에의 경로값에 없는 함수는 현재 있는 디렉토리를 검색을 하고 그래도 없으면 함수를 찾을 수 없다는 에러 메시지를 띄우게 된다. 그러므로 average.m이라고 바꾼 파일을 현재 작업하고 있는 디렉토리 내에 저장을 하던가 아니면 matlab의 환경변수 path내에 선언이 되어 있는 임의의 디렉토리에 저장을 해야된다. 기본적인 환경에서는 `c:\MATLABR11\work` 내에 저장하는 것이 일반적인 경우이다.

자 다시 위의 함수를 분석해 보자. 입력 값으로 x를 사용하였다. 이렇게 x라고 정의를 한 값은 이 값이 벡터가 될지 아니면 하나의 변수가 될지는 정하지 않은 것이다. 하여간 단순히 난 x라는 변수 이름으로 입력값을 사용하겠다고 정한 것이다. 여기서 matlab에 대한 얘기를 한가지 더 해야되는데 변수의 영향이 어디까지 미치는 가에 대한 것이다. 즉, local variable과 global variable이다. 지역 변수라고 하는 것은 일반적으로 함수가 종료가 되면 더 이상 값을 가지고 있지 않다. 그렇지만 전역 변수를 프로그램이 종료될 때까지 값을 가지고 있다. 함수 내에 정의된 변수는 특별한 지시자가 없는 경우에는 지역변수이다. 전역변수로 사용하기 위해서는 global이라는 지시자를 사용한다. 일단 다음과 같은 함수를 만들자.

```
function testout=test(xx)
global a b
a=10;
b=20;
c=30;
testout=30;
```

그 다음 다음과 같이 실행을 한다.

```
>>global a b
>>a=1;
>>b=2;
>>c=3
>>test(c);
>>a

a =
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      10

>>b

b =

      20

>>c

c =

       3
```

이 예제에서 보면 a, b, c라는 변수를 메인문에서 사용했으며 a와 b는 각각 global로 선언을 하였다. 각각에 1과 2라는 값을 할당을 하고 c라는 지역변수에는 3이라는 값을 할당했다. test라는 함수를 실행한 후 다시 각각의 변수에 어떠한 값이 있는지 확인을 해 보았다. 결과는 a와 b 변수는 값이 바뀌었으나 c 변수값은 바뀌지 않았다. 자 그러면 test라는 함수를 한번 보자. 이 함수는 a, b, c, xx, testout 라는 여러가지 변수를 사용했지만 입력값으로 xx라는 변수를 사용했고 출력값으로 testout이라는 변수를 사용했다. 그렇지만 xx라는 변수는 사용되지 않았으며 a와 b는 단순히 global로 선언이 되어있고 그 값은 정해진 값인 10과 20으로 치환이 되어있다. c라는 변수도 사용이 되었지만 이 값은 지역변수 즉, test라는 함수내에서만 유효한 변수이며 값이 30이라고 치환되었지만 메인 함수의 값에는 전혀 영향을 미치지 못했다.

다시 함수의 구조에 대해서 보기로 하자. 일단 x라는 변수가 들어오면 변수의 크기를 구하는 size라는 함수를 이용해서 이 변수의 크기를 구한다. 그 크기에 대한 m과 n라이는 변수에 대해 검사를 해보는데 현재 계산하고자 하는 것은 벡터로 주어진 변수의 평균값을 계산하는 것이므로 변수가 행렬의 형태로 들어오거나 아니면 단순한 상수로 입력되는 경우 에러 메시지를 보여주고 더 이상 진행을 하지 않는다. error 함수에 대한 것은 help error를 해서 확인해 보기 바란다. 그 다음 주어진 변수가 벡터로 주어진 경우에는 변수의 모든 요소의 합을 계산하는 sum이라는 함수를 이용하고 변수의 길이($1 \times n$ 이거나 $n \times 1$) 가운데 가장 큰 값을 찾는 length함수의 결과 값을 이용해서 평균을 구한다. 그렇게 얻어진 값을 mean_value 라는 출력 변수로 넘긴다.

마지막으로 사실 matlab에는 이미 평균을 구하는 루틴이 있다. mean이라는 함수를 이용하는 경우 주어진 변수가 행렬로 되어 있던 아니면 벡터로 되어 있던 그 값을 계산할 수 있다. 행렬로 되어있는 경우에는 열을 기준으로 하던가 아니면 행을 기준으로 각각의 평균값을 계산할 수 있다. 자세한 내용은 help mean을 참고 하기 바란다. 그렇지만 위에서 만든 average는 여러분들에게 함수를 작성하는 방법을 설명하기 위해서 만든 함수이다. matlab 프로그램을 좀더 작성함에 따라 여러가지 함수를 작성해야만 된다. 또한 프로그래밍을 하는 관점에서도 되도록이면 간단한 프로그램을 작성하기 위해서 메인문에서 모든 계산과정을 진행하는 것

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

보다는 각각의 작은 함수로 작성을 해서 계산하는 것이 더 좋은 프로그래밍 기법이다.

A.5 Matlab에서 그림 그리기

이번 장에서 matlab에서 나온 결과를 그림으로 그리는 방법에 대해서 생각을 해보자. 아무리 좋은 결과가 얻어졌다고 하더래도 그 결과를 사람들이 알아보기 좋게 보여주지 않으면 관심을 얻기 힘들다. 자 그러면 이제 그림을 그리는 방법에 대해서 생각을 해보자.

A.5.1 plot 명령어

사실 대부분의 2차원 그래프나 그림은 plot이라는 명령어로서 그릴 수 있다. 그럼 일단 help plot으로 얻어지는 것을 가지고 얘기를 해 보자.

```
>>help plot

 PLOT   Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, length(Y)
    disconnected points are plotted.

    PLOT(Y) plots the columns of Y versus their index.
    If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
    In all other uses of PLOT, the imaginary part is ignored.

    Various line types, plot symbols and colors may be obtained with
    PLOT(X,Y,S) where S is a character string made from one element
    from any or all the following 3 colunms:

           y     yellow        .     point              -     solid
           m     magenta       o     circle             :     dotted
           c     cyan          x     x-mark             -.    dashdot
           r     red           +     plus               --    dashed
           g     green         *     star
           b     blue          s     square
           w     white         d     diamond
           k     black         v     triangle (down)
                               ^     triangle (up)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
                                        <     triangle (left)
                                        >     triangle (right)
                                        p     pentagram
                                        h     hexagram

    For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus
    at each data point; PLOT(X,Y,'bd') plots blue diamond at each data
    point but does not draw any line.

    PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
    the (X,Y,S) triples, where the X's and Y's are vectors or matrices
    and the S's are strings.

    For example, PLOT(X,Y,'y-',X,Y,'go') plots the data twice, with a
    solid yellow line interpolating green circles at the data points.

    The PLOT command, if no color is specified, makes automatic use of
    the colors specified by the axes ColorOrder property.  The default
    ColorOrder is listed in the table above for color systems where the
    default is yellow for one line, and for multiple lines, to cycle
    through the first six colors in the table.  For monochrome systems,
    PLOT cycles over the axes LineStyleOrder property.

    PLOT returns a column vector of handles to LINE objects, one
    handle per line.

    The X,Y pairs, or X,Y,S triples, can be followed by
    parameter/value pairs to specify additional properties
    of the lines.

    See also SEMILOGX, SEMILOGY, LOGLOG, GRID, CLF, CLC, TITLE,
    XLABEL, YLABEL, AXIS, AXES, HOLD, COLORDEF, LEGEND, and SUBPLOT.

>>
```

이 명령어를 잘 보면 일단 기본적인 것을 그릴때는 단순히 `plot(x,y)`라고 해 주면 된다. 그 다음 여러가지를 꾸미고 싶은 경우 더 많은 옵션을 사용하면 된다. 예제를 보면서 하나씩 만들어 보자.

```
>>a=[1 2 3];
>>b=[2 3 4];
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
>>plot(a,b);
```

자 이 명령어에 의해 만들어진 것을 한번 보자. 단순히 그냥 $y = x + 1$ 형태의 함수를 그렸다

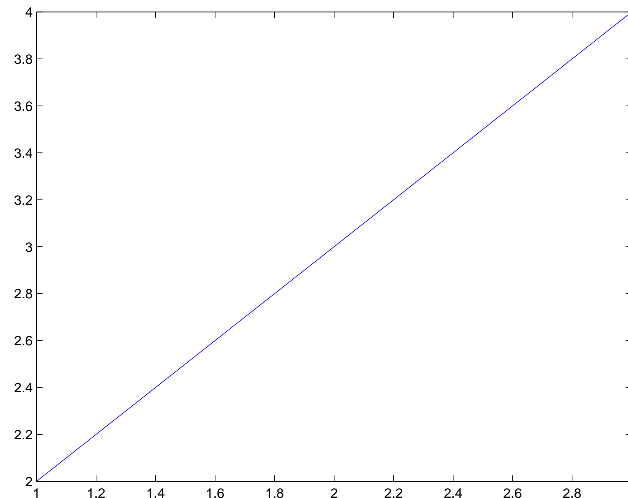

Figure A.1: Simple plot example: plot(a,b)

고 볼 수 있다. 그렇지만 이번 경우는 사실 이러한 함수를 그린 것이 아니고 단순히 점을 그릴려고 했던 것이기 때문에 다음과 같이 수정을 해야된다.

```
>>plot(a,b,'.');
```

그림에서 보이듯이 알아보기가 좀 힘들다. 그렇다면 다음과 같이 바꿔보자.

```
>>plot(a,b,'o');
```

이제 좀전보다 좀더 나아진거 같은데 이번엔 색을 바꿔보자.

```
>>plot(a,b,'mo');
```

이렇게 하면 색깔이 바뀐다. 그러면 이번에는 각 포인트들 사이에 들어갈 부분에 대해서 생각을 해보자. 점선으로 각 포인트를 연결해보자.

```
>>plot(a,b,':o');
```

기본적으로 색에 대한 것이 없는 경우에는 처음에 파란색을 사용한다. 그러면 이번에는 색도 바꾸고 포인트를 표시하고 그리고 포인트 사이에 직선을 넣어보자. 각각의 옵션을 더해주면 된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

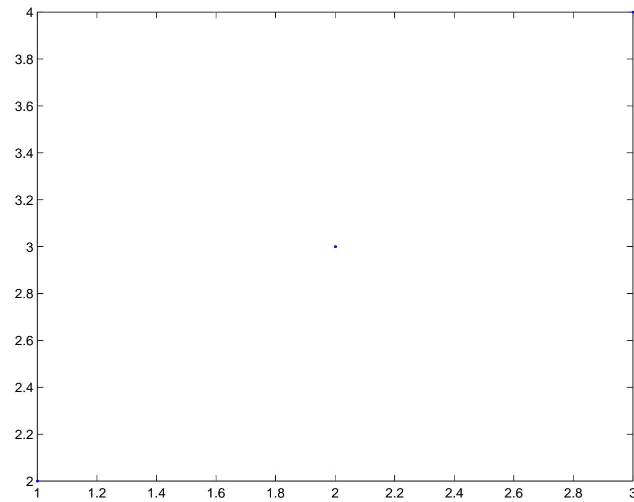

Figure A.2: Simple plot example: plot(a,b,'.')

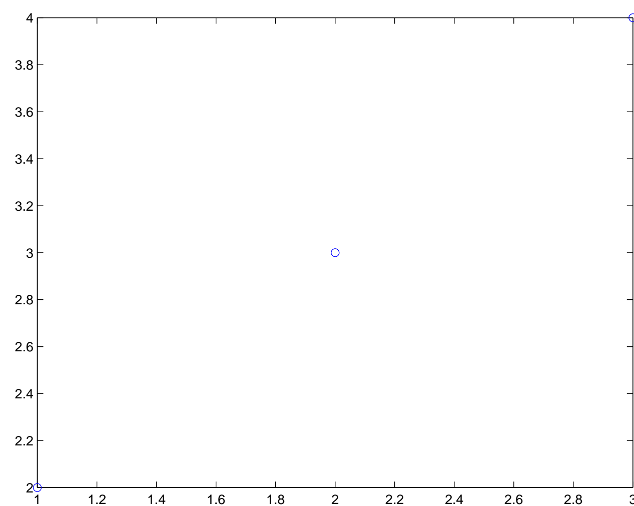

Figure A.3: Simple plot example: plot(a,b,'o')

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

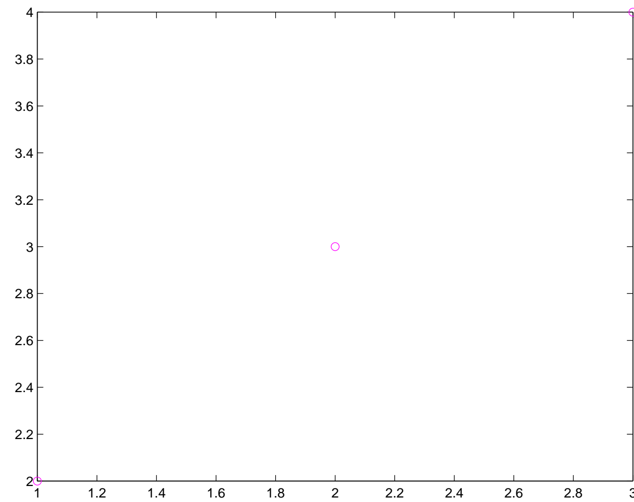

Figure A.4: Simple plot example: plot(a,b,'mo')

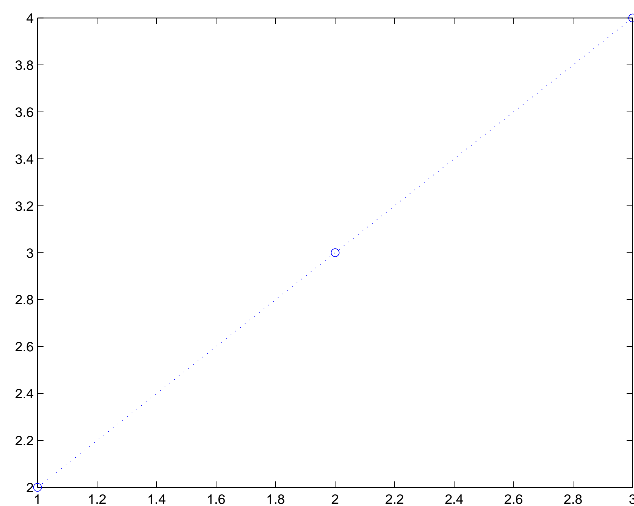

Figure A.5: Simple plot example: plot(a,b,':o')

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

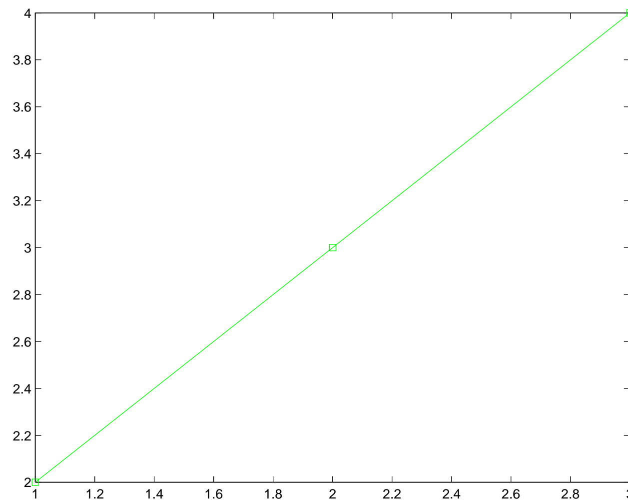

Figure A.6: Simple plot example: plot(a,b,'g-s')

```
>>plot(a,b,'g-s');
```

이렇게 하면 이제 녹색줄이 그어지고 그리고 각 포인트를 네모로 표시하였으며 포인트 사이에는 직선을 그었다. 그런데 한번 더 생각을 해보자. 그래프의 모습이 좀 답답해 보이지 않는가? 아무리 사용자가 1부터 3까지의 데이터만을 넣었다고 하더래도 좀 넉넉하게 보이는 것이 좋지 않을까? 그러면 즉, x축의 범위를 정해줘야 된다. 그걸 해 보자.

```
>>plot(a,b,'vr-.');
>>axis([0.0 4.0 0.0 5.0])
```

자 이렇게 하면 좀더 보기가 좋을것이다. 그런데 아직도 뭔가 허전하다. x축 값은 무엇이고 y축 값은 무엇이라고 할때 어떻게 해야될까? 그건 xlabel과 ylabel이라는 것을 이용하면 된다.

```
>>plot(a,b,'vr-.');
>>axis([0.0 4.0 0.0 5.0])
>>xlabel('X data'); ylabel('Y data');
```

그럼 이제 간단한 그림의 마지막으로 그림의 제목을 달아보자. 이것은 간단히 title이라는 것을 이용하면 된다.

```
>>plot(a,b,'vr-.');
>>axis([0.0 4.0 0.0 5.0])
>>xlabel('X data'); ylabel('Y data');
>>title('Test graph')
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

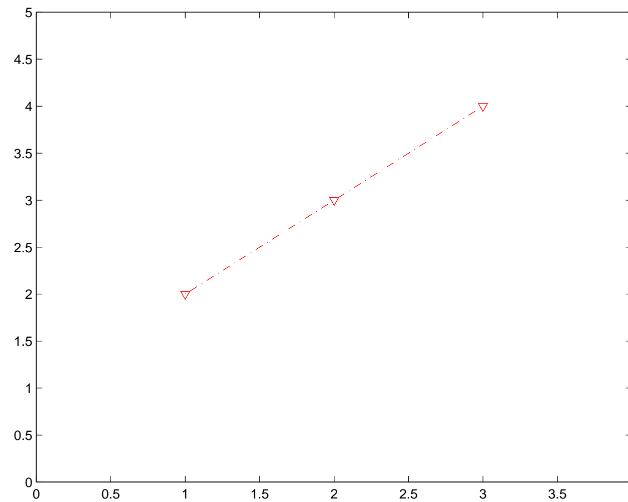

Figure A.7: Simple plot example: plot(a,b,'vr-.'), axis([0.0 4.0 0.0 5.0])

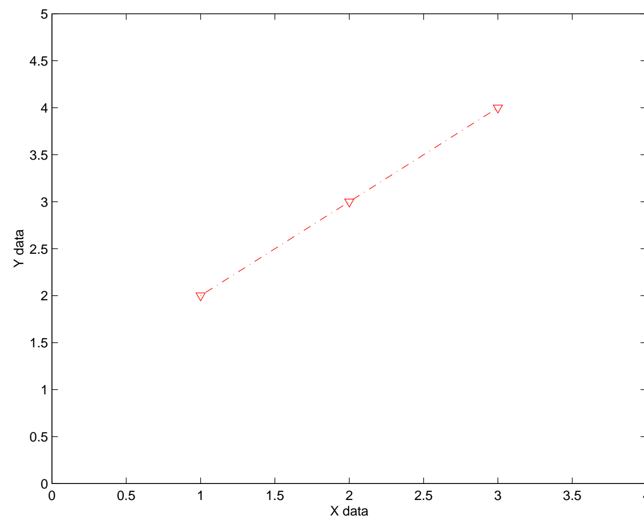

Figure A.8: Simple plot example: xlabel('X data'), ylabel('Y data')

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

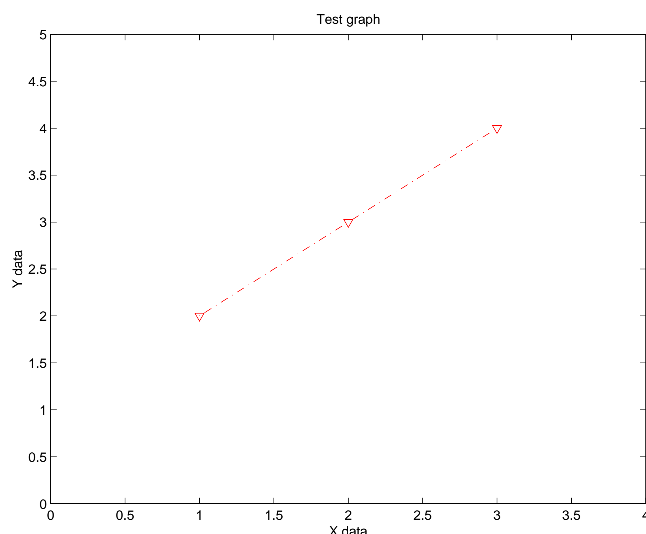

Figure A.9: Simple plot example: title('Test graph')

이제는 아마도 기본적인 그래프를 그리는 방법은 배웠을 것이다. 그러면 이걸 프린트 해보자. 물론 그냥 프린트를 하면 여러분의 프린터에 프린트가 된다. 그러면 파일로 저장할려면 어떻게 해야될까 ? 그것 또한 print라는 명령어를 이용하면된다. 그런데 정말 print는 복잡한 옵션을 가지고 있다. 확인하고 싶으면 help print해 봐라. 그 가운데 간단한 몇가지 옵션만 설명하겠다.

```
>>print -dtiff
```

위와 같이 사용하면 tiff 포멧으로 그림을 만드는데 정확히 정해진 그림은 없다 또한 파일도 없다. 그러므로 어떻게 되냐면 일단 현재 그림 가운데 활성화 된 것이 선택된다. 즉, 그림이 여러가지 있는 경우에는 가장 마지막에 만들어졌거나 아니면 여러분이 여러가지 그림 가운데 마우스로 선택을 했거나 아니면 뭐 어떻게 됐던지 간에 그림 가운데 활성화를 시킨 그림을 그냥 파일로 저장해 버린다. 아마 여러분의 작업 디렉토리에는 figure1.tif 파일이 만들어져 있을 것이다.현재 지금 활성화 되어있는 그림이 첫번째 그림이라면 figure라는 이름 뒤에 그림의 숫자를 붙여서 만들어 준다. 여러분이 좀 기억해야 될 것이 있는데 이것은 matlab과 상관 없이 컴퓨터에서 그림 파일을 이용할때에는 bmp 형식의 그림을 만들면 안된다. 왜냐면 이파일은 무지 크기 때문에 어떤경우에는 수십메가가 될 수도 있다. 그러니 꼭 tif나 jpg 아니면 gif 형식의 그림을 사용하기 바란다. 그리고 matlab에서 이러한 그림의 형식을 지정해주는 옵션은 -d라는 옵션뒤에 써주면 된다. tiff나 jpeg, png 등이 있으니 주로 사용하게 될 것은 -dtiff나 -djpeg 등이 될 것이다. 자 그러면 좀더 옵션을 사용해 보자. 그림이 여러개 있는 경우에 내가 원하는 그림만을 파일로 저장하기 위해서는 어떻게 할까 ?

```
>>print -dtiff -f3
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

-f라는 옵션뒤에 그림의 숫자를 붙여주면 된다. 그러면 그림 숫자에 맞는 그림을 figure3.tif 형식으로 만들어 준다. 그러면 이제 그림의 이름을 정해보자.

```
>>print -dtiff -f3 app3
```

이렇게 명령어를 입력하면 app3.tif라는 파일을 만들어준다. 자 이제 그림을 만들고 그리고 저장하는 방법에 대해서는 배웠다. 좀더 고급 단계로 나가보자.

A.5.2 고급 plot 명령어

이번에는 좀더 plot 명령어에 대해서 알아보자. plot 명령어를 잘 사용하면 굳이 다른 그래프 프로그램을 거의 쓸 필요가 없다. 필자의 경우에는 해외저널에 제출하는 논문의 경우에도 그냥 plot을 이용해서 그린 그림을 제출하고 있다. 그런데 저번에 제출한 논문의 경우 출판사에서 수정을 아예하지 않았다. 자 그럼 시작해 보자.

한 그래프에 두개 이상의 데이터 셋을 찍고 싶을때는 어떻게 할까? 예제를 보자. 일단 찍을 데이터를 만들기 위해서 다음의 프로그램을 만든 후 gen_data.m으로 저장을 하자.

```
j=0;
for i=0:0.1:1
   j=j+1;
   a(j) = i^2;
   b(j) = i^3;
   k(j) = i;
end
```

그 다음 다음과 같이 위의 프로그램으로 데이터를 만들고 그림을 그려보자.

```
>>gen_data
>>plot(k,a,k,b);
```

그림을 보면 두개의 결과가 출력되어 있다. 출력을 위해 만든 데이터는 x^2과 x^3의 값을 x가 0부터 1까지 0.1 간격으로 계산을 해서 만든 것이다. 두개의 결과가 있는데 기본적인 경우에는 이 가운데 파란색이 첫번째 데이터이고 두번째 것은 녹색으로 만들어진다. 그러므로 두개의 결과를 출력하기 위해서는 두개의 출력 데이터 셋이 x축과 y축에 대해 필요하게 된다. 다음에는 이 두개의 결과를 나눠보자. 단순하게 두개의 그래프에 나눠서 출력할 수도 있다. 즉,

```
>>gen_data
>>plot(k,a);
>>plot(k,b);
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

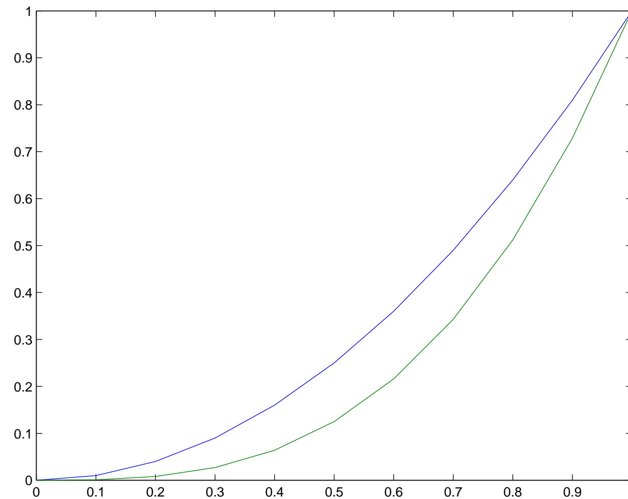

Figure A.10: Simple plot example: plot(k,a,k,b)

그렇지만 만약 위와 같은 명령을 입력한다면 그림이 두개로 나눠져서 그려지는 것이 아니라 최종으로 만든 것만 남는다. 그러므로 그림을 두개 그릴려면

```
>>gen_data
>>figure, plot(k,a);
>>figure, plot(k,b);
```

figure라는 명령어를 이용해서 그림을 그릴 수 있는 공간을 만들고 그려야만된다. 이 명령어를 이용하면 만들어지는 그림의 번호를 지정할 수도 있다.

```
>>gen_data
>>figure(3), plot(k,a);
>>figure(5), plot(k,b);
```

위와 같이 번호를 명시해주면 만들어지는 그림의 번호를 명시할 수도 있다. 자 그런데 현재 만들고자 하는 그림은 위와 같은 것이 아니라 한 그림내에 두개의 그래프를 그리는 것이다. 이렇게 하기 위해서는 subplot이라는 명령어를 사용한다. 이 명령어는 세개의 값을 인수로 받는다.

```
subplot(no_row, no_col,position_figure);
```

좀 복잡해 보이는데 나눠질 그림의 갯수를 행과 열의 갯수를 정해서 만들고 그리고 난 후 하나씩 만들 그림의 위치를 정해주면 된다. 예제를 보면서 만들어 보자.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
>>gen_data
>>subplot(2,1,1), plot(k,a);
>>subplot(2,1,2), plot(k,b);
```

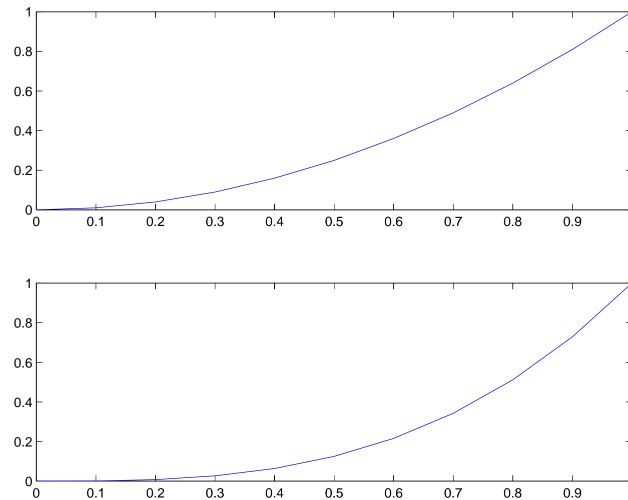

Figure A.11: Simple plot example: subplot 2X1

그림에서 보여지는 것과 동일하게 일단 행은 두개로 만들고 열을 하나로 만든 후에 첫번째 것은 1행1열의 그림이고 두번째 것은 2행1열의 그림이 된다. 그러면 테스트 용으로 2개의 데이터를 더 추가해보자. 이 데이터는 $-x^2+1$과 $-x^3+1$의 결과이다.

```
j=0;
for i=0:0.1:1
   j=j+1;
   a(j) = i^2;
   b(j) = i^3;
   c(j) =-i^2+1;
   d(j) =-i^3+1;
   k(j) = i;
end
```

전체 그림을 하나에 그리면 이것을 다음의 subplot 명령어를 하나씩 사용해 가며 바뀌어지는 것을 관찰해보자.

```
>>subplot(2,2,1),plot(k,a)
```

처음의 결과가 왼쪽 위에 생겼다. 그 다음

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

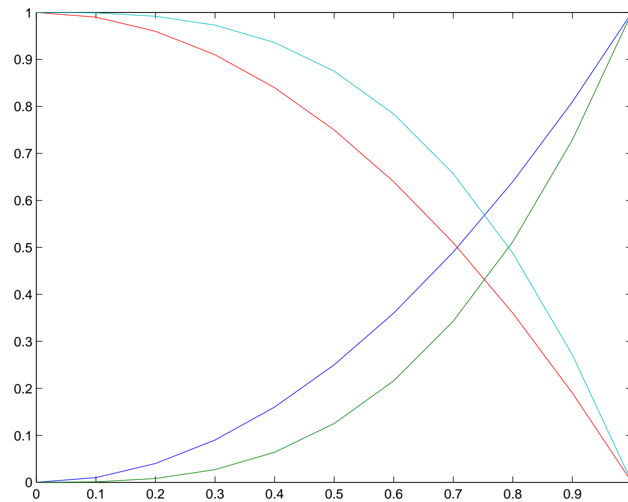

Figure A.12: Simple plot example: plot all data

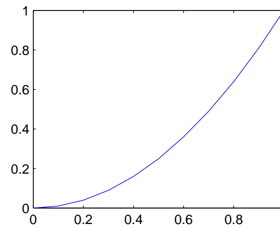

Figure A.13: Simple plot example: subplot(2,2,1), plot(k,a)

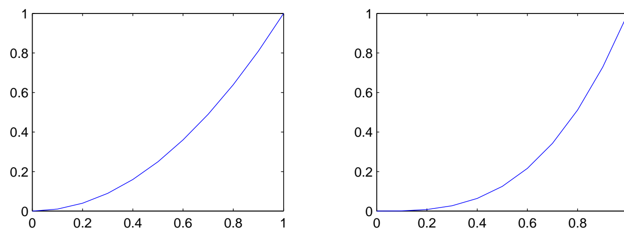

Figure A.14: Simple plot example:subplot(2,2,2), plot(k,b)

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
>>subplot(2,2,2),plot(k,b)
```

이번에는 오른쪽 위에 그림이 그려진다. 계속 해보면

```
>>subplot(2,2,3),plot(k,c)
```

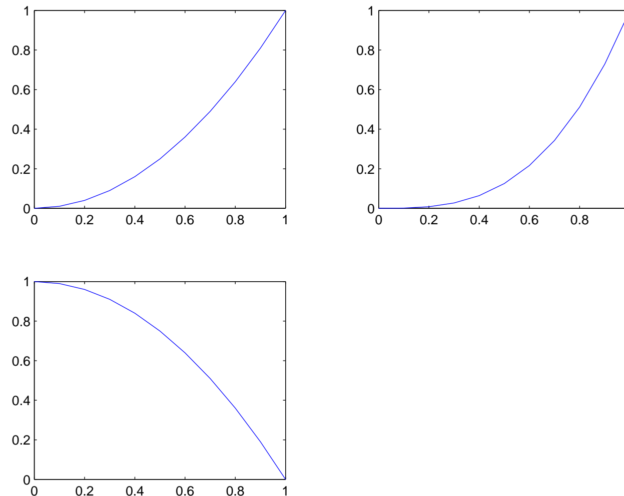

Figure A.15: Simple plot example: subplot(2,2,3), plot(k,c)

이번에 왼쪽 밑에 그렸고 마지막으로 오른쪽 밑에 그림을 그리기 위해서는

```
>>subplot(2,2,4),plot(k,c)
```

자 이제 그림이 그려지는 순서를 알 수 있겠는가 ? 하나씩 순서대로 만들어진다.

여러개의 그림을 그리는 방법을 알았으니 이번에는 좀더 다른 그림을 그려보자. 좀전에 그린 그림 A.5.2의 경우에 어떤 그림이 어떤 것을 나타냈는지 알기 어렵다. 물론 이전에 첫번째 데이터는 파란색으로 그리고 그 다음 녹색으로 그린다고 했으니 몇가기를 구분할 수 있겠지만 그리 쉬운 것이 아니다. 이러한 경우에는 legend를 달아서 알기 쉽게 만들수 있다. 사용 방법은 다음과 같다.

```
legend('legend1','legend2',....'Pos')
```

각각의 데이터에 대한 범례를 표시한다음 마지막으로 이 범례를 어디에 나타낼지를 결정해 주면 된다. 각각의 위치에 대한 설명은 다음과 같다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

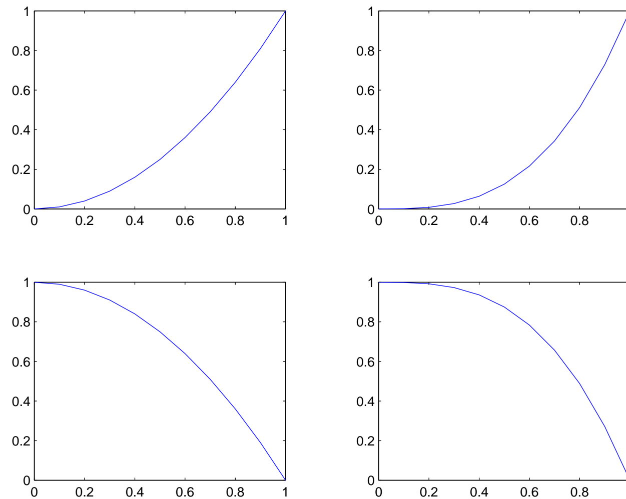

Figure A.16: Simple plot example: subplot(2,2,4), plot(k,d)

```
location:
          0 = Automatic "best" placement (least conflict with data)
          1 = Upper right-hand corner (default)
          2 = Upper left-hand corner
          3 = Lower left-hand corner
          4 = Lower right-hand corner
         -1 = To the right of the plot
```

이제 감이 좀 잡히는가? 자 그럼 좀전에 만든 그림에 각각의 범례를 달아보자.

```
>>plot(k,a,k,b,k,c,k,d);
>>legend('x^2','x^3','-x^2+1','-x^3+1',0)
```

이렇게해서 그린 그림을 보자. 보면 범례가 붙어 있는 부분이 사실 좀 마음에 안든다. 왜 matlab이 여기에 붙였을까? 잘 보면 왼쪽이던지 오른쪽이던지 막상 붙여둘 곳이 없다. 그러므로 matlab이 봤을때는 가장 좋은 부분이 가운데쯤 될꺼라 생각하고 붙인것이다. 그러면 이것을 수정하기 위해서는 그냥 그림에서 범례가 붙은 부분을 마우스로 끌어서 옮기면 된다. 그렇지만 아마 적당한 곳을 찾기 힘들꺼다. 그리고 다시 잘 보면 범례에서 내가 쓴 글들이 수식으로 표현되었다. 몇가지는 가능하니 참고하기 바란다.

이번에는 추가적 기능으로 grid를 만들어보자. 이것은 그냥 `grid`라고 입력하면 된다.

자 이제 `fplot`이라는 것에 대해서 알아보자. 위의 예제 가운데 함수를 값을 그리고 위해서 우리는 계산을 할 x 값을 적당히 나눠서 설정을 해 주고 난 후 그 값을 이용해서 계산을 해서 함수의 그래프를 그렸다. 물론 복잡한 경우에는 그렇게 할 수 밖에 없다. 그렇지만 이것을 한번 생각해보자. 왜 함수를 그려주는 간단한 방법은 없을까? 있다. 그것이 `fplot`이다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

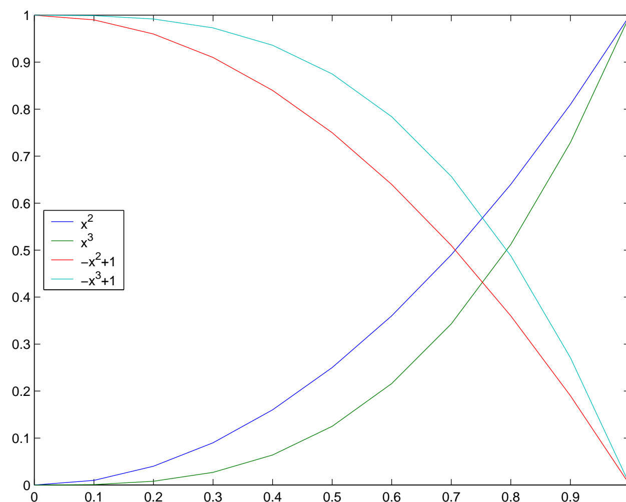

Figure A.17: Simple plot example: legend

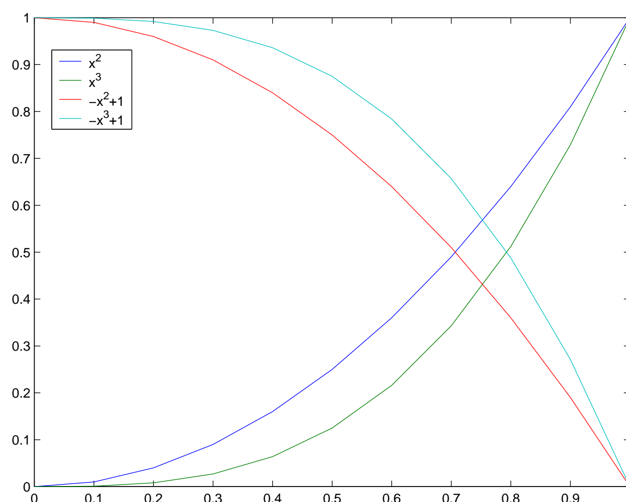

Figure A.18: Simple plot example: legend with user defined position

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

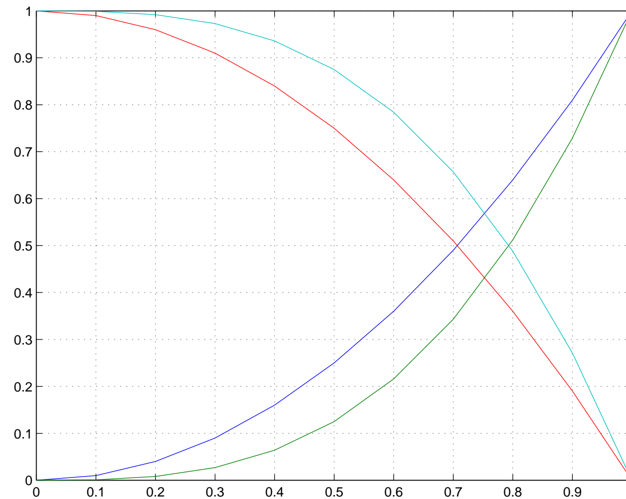

Figure A.19: Simple plot example: grid

```
FPLOT(FUN,LIMS)
```

어떻게 보면 간단해 보이는데 문제는 FUN을 만드는 것이다. 이 부분은 함수를 만드는 것과 동일하다. 자 일단 x^2이라는 함수의 값을 0부터 2까지 그려보자.

```
function f=x2(x)
f=x^2;
```

다음과 같이 만든 파일을 일단 x2.m이라고 저장을 하고난 후 다음과 같은 내용을 입력하면 그림을 그릴 수 있다.

```
>>fplot(x2,[0 2])
```

그런데 이렇게 함수까지 만들어서 사용하는 것은 좀 번거럽다. 단순한 함수의 값을 그려보기 위해서는 위와 같이 굳이 그리고자 하는 함수를 따로 지정할 필요가 없다. 간단히 다음과 같이 입력하면된다.

```
>>fplot('x^2',[0 2])
```

그러므로 위에 4개의 그래프를 그리는 예제를 바꿔서 그려보면 다음과 같이 입력하면 된다. 다음과 같이 입력해도 충분히 그림을 그릴 수 있다.

```
>>subplot(2,2,1), fplot('x^2',[0 1])
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

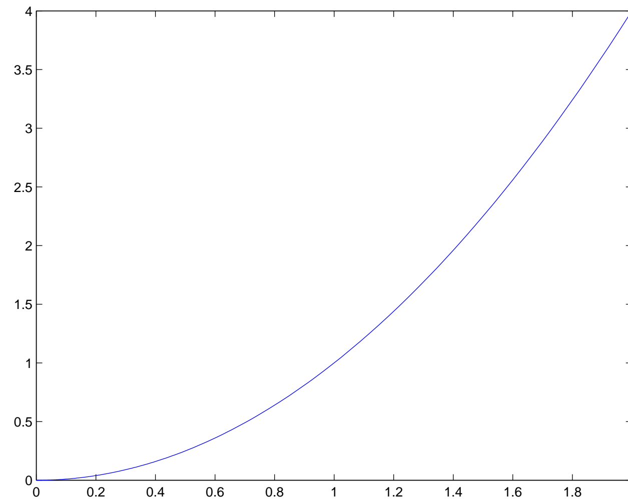

Figure A.20: Simple plot example: fplot

```
>>subplot(2,2,2), fplot('x^3',[0 1])
>>subplot(2,2,3), fplot('-x^2+1',[0 1])
>>subplot(2,2,4), fplot('-x^3+1',[0 1])
```

이번에는 다음과 같은 상황을 생각해 보자. 계산 도중에 어떤 그림을 그리고자 그림을 그렸다고 하자. 그러다가 또 다른 계산을 하다가 그림을 그릴때는 어떻게 해야될까 ? 그냥 그리면 된다고 ? 그럼 그림이 없어진다니까 !

```
>>fplot('x^2',[0 2])
>>fplot('x^2',[0 3])
```

이렇게 입력을 하면 마지막 그림인 x^3의 그림만 남아있게 된다. 그러면 어떻게 하면 하나의 그림에 두개의 결과를 넣을 수 있을까 ? 물론 따로 저장을 한 다음 그리면 되겠지만 그것말고 방법은 없을까 ?

```
>>fplot('x^2',[0 2])
>>hold on
>>fplot('x^2',[0 3])
```

위와 같이 hold라는 명령어를 사용하면 한 그림안에 두개의 그래프를 그릴 수 있다. hold라는 명령어는 현재 그림에 그려진 그래프를 유지하라는 명령어이다. 이 상태에서 원하는 다른 그림을 그리면 된다. 그리고 해제하기 위해서는 `hold off`해주면 된다. 주의해야 될 것은 두 그래프 모두 독립적이기 때문에 모두 파란색으로 그려졌다는 것이다. 특별히 지정을 해 주지

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

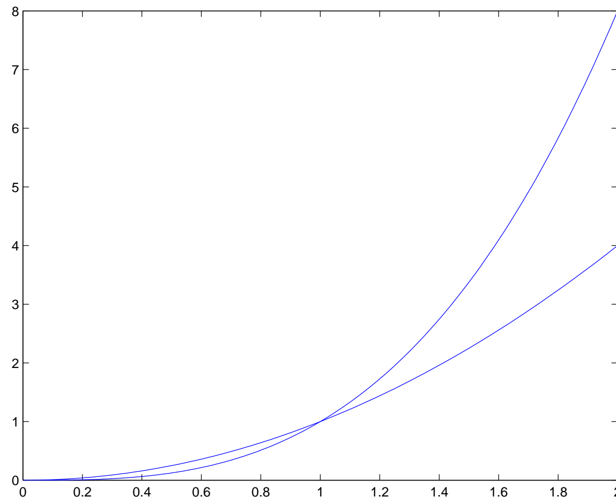

Figure A.21: Simple plot example: hold

않는 경우에는 그래프를 그리는 경우 첫번째 그래프는 파란색으로 그린다는 것은 이미 얘기를 했다.

자 이제 마지막으로 그림을 지울려면 어떻게 해야될까? 물론 윈도우를 닫으면 된다. 그렇게 이렇게 하는 경우는 그림을 지운 것이 아니고 정확하게는 그림창을 닫아버린 것이다. 이렇게 하지말고 그냥 matlab의 명령창에서 할 수 있는 방법은 없을까? 예전에 데이터를 지우는 것은 `clear`라는 명령어를 사용한다고 했었다. 이와 비슷하게 현재 그림창의 그래프를 지우기위해서는 `clf`라는 명령어를 사용하면 된다. 또한 그림창을 닫기 위해서는 `close` 명령어를 사용하면 된다. 게다가 그림창 모두를 닫기 위해서는 `close all`이라고 명령어를 입력하면 된다. 그림창을 만드는 명령어는 이미 얘기를 했다. `figure`다.

A.5.3 그림을 그리는 다른 명령어들

지금까지는 단순히 대부분 plot이라는 명령어를 이용해서 그림을 그리는 방법에 대해서만 배웠다. 사실 matlab에서 plot은 기본적인 명령어일 뿐이다. 이것외에도 상당히 많은 명령어들이 있다. 결론적으로 대부분의 그림은 모두 matlab에서 그릴 수 있다.

일단 다음의 데이터 셋을 그림으로 그려보자. 일단은 그냥 plot을 이용해서 그려보자.

```
>>t=[0 1 2 3 4];
>>y=[0 1 1 1 1];
>>plot(t,y), axis([0 4 0 1.1])
```

그림을 좀 정확하게 보기위해서 축의 범위를 약간 조절했다. 결과 그래프를 보면 사실 데이터 셋에서는 0부터 1사이에 어떠한 값을 가진다는 것을 명시하지 않았다. 그러나 그래프는

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

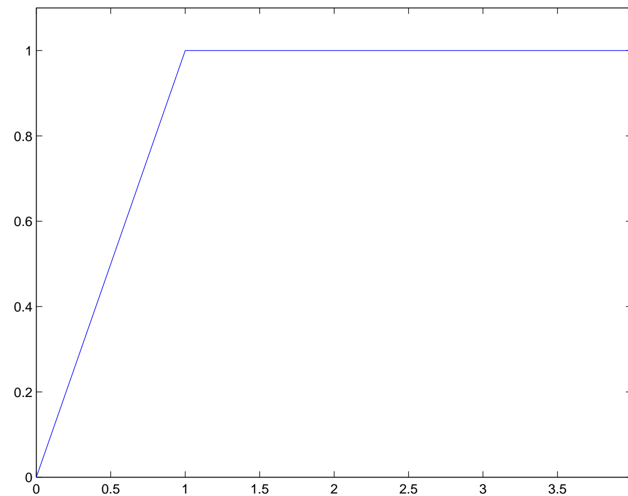

Figure A.22: Advanced plot example: simple plot

결과 그림을 그 사이의 값은 두 지점을 연결해서 만들었다. 그런데 이게 우리가 의도한 바하고 다른 경우에는 어떻게 출력을 해야할까 ? 즉, 우리가 원하는 그림은 처음에는 0의 값을 가지고 있다가 갑작스래 1인 값에서 y의 값이 어떠한 값을 가지도록 하기위해서는 어떻게 해야될까 ? 이런경우에는 기존에 사용해오던 plot으로는 아무리 해도 깨끗하게 그릴 수 없다. stairs라는 명령어를 사용해야된다. 이 명령어는 이름에서도 알 수 있겠지만 계단형의 결과를 잘 그려주는 명령어이다. 사용 방법은 plot과 유사하다.

```
>>t=[0 1 2 3 4];
>>y=[0 1 1 1 1];
>>stairs(t,y), axis([0 4 -0.1 1.1])
```

결과 그림에서 보이듯이 원하는 바를 얻을 수 있다. 주로 이러한 그림은 제어를 하는 경우 그려야될 필요가 상당히 많다. 대학원 과정에서 배우게 될 zero-order hold의 경우에 많이 쓰게 될 것이다.

이번에는 두개의 y축 값을 가지고 있는 그림을 그려봅시다. 가끔 그림을 그리다가 보면 y축의 스케일이 다른 경우가 있습니다. 이런 경우는 어떻게 그려야 할까요 ? `plotyy`라는 명령어를 사용하면 됩니다. 예제로서 좀전에 사용했던 x^2과 x^3 그림을 그려봅시다. 만약 이걸 그냥 한 그래프에 그리면 다음과 같을 것입니다.

```
>>t=0:0.1:2;
>>y1=t.^2;
>>y2=t.^3;
```

데이터를 만들기위해서 사용한 ‘ `.^` ’ 부분은 행렬부분에서 이미 설명을 했다. 각 요소에 대한 제곱을 계산하는 경우에 사용한다. 그러므로 y1과 y2는 각각 t^2과 t^3 값을 계산하는 것이

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

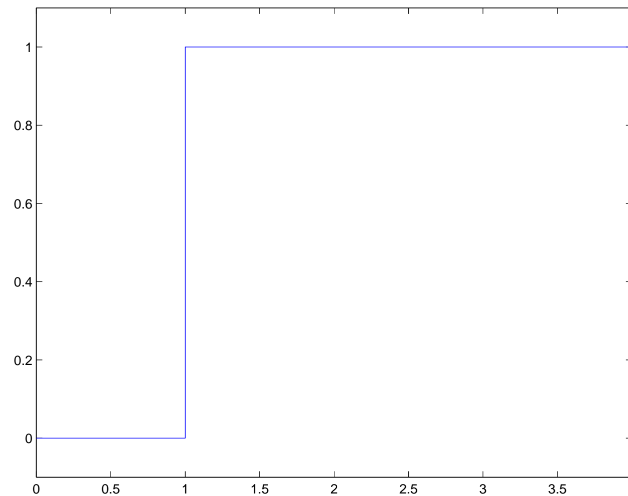

Figure A.23: Advanced plot example: stairs plot

다. 이것을 그냥 그래프에 그리는 경우에는 다음과 같은 결과를 얻는다. 이 그래프에서는 y축

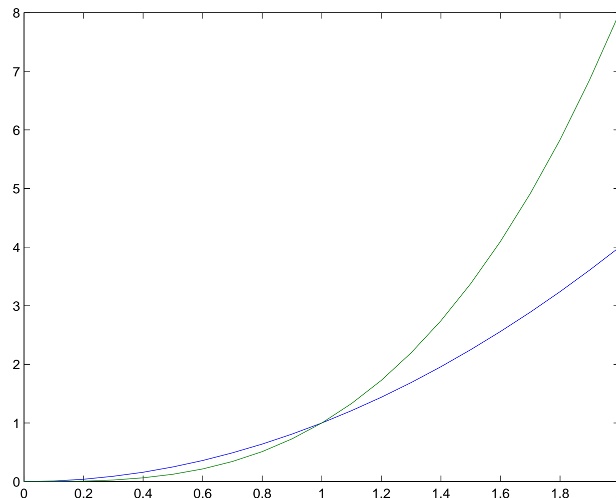

Figure A.24: Advanced plot example: simple plot

의 값이 하나이기 때문에 결과 그림에서 서로의 값을 비교하는 경우에는 좋겠지만 그렇지 않고 두 그래프가 정확히 어떤 값을 나타내고 있는 가를 알기 위해서는 썩 좋지 않다. 그렇지만 이렇게 y축을 두개 사용하는 경우에는 두 그래프의 값이 정확히 어떤 값을 가르키고 있는지 알 수 있다. 또한 그림이 그려진 순서는 파란색과 녹색으로 그려지며 축의 값도 다르다는 것을 알 수 있다. 이렇게 사용하는 것 외에 그림의 형식을 각각 다르게 사용할 수 있는데 위의 예제는 모두 plot을 사용한 것이고 이것외에 다른 그림을 그릴 경우에는 그 그림의 형식을 명시 해 줘야만 한다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

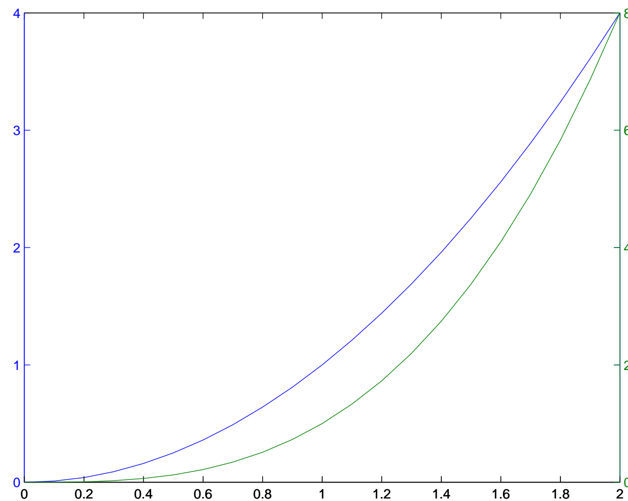

Figure A.25: Advanced plot example: plotyy

```
>>t=[0 1 2 3];
>>y=[0 1 1 1];
>>plotyy(t,y,t,y,'plot','stairs')
```

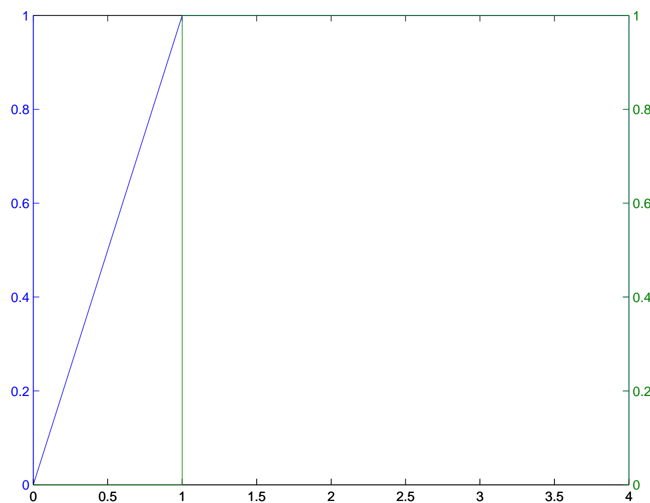

Figure A.26: Advanced plot example: plotyy with different type plot

이렇게 하면 하나는 그냥 plot으로 그림을 그리고 나머지 하나는 stairs 방식으로 그림을 그린다. 이러한 것은 다양한 예에 적용할 수 있다. 좀더 이 부분에 대해서 논의를 해야되는데 일단 접어두고 graph에 대해 좀더 심도 있게 배워보자.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

현재까지는 간단히 그림을 그리는 방법에 대해서 논의를 했는데 이것보다 좀더 복잡하게 그림을 제어할 수 있는 방법에 대해서 약간 논의를 하겠다. 이유는 `plotyy`의 경우에서 각각의 y축에 라벨을 붙이기 위해서는 그냥 `ylabel`이라는 명령어를 사용해서는 어디에 라벨을 붙여야 될지 모르기 때문에 두개의 y축을 명시해 줘야 된다. plot이라는 명령어의 help 내용을 보면 마지막에 이러한 얘기가 나온다.

```
PLOT returns a column vector of handles to LINE objects, one
handle per line.
```

이 얘기가 무엇을 얘기하는지 간단히 알아보자. 일단 어떠한 함수나 계산 결과를 얻을 때 다음과 같이 사용한다.

```
>>5

ans =

     5

>>a=5

a =

     5

>>
```

이와 같은 결과를 보면 그냥 5라는 값을 넣으면 matlab에서는 기본적으로 리턴을 해서 저장할 변수가 없는 경우에는 단순히 ans라는 변수에 5라는 값을 저장하고 5라는 값을 echoing한다. 그런데 a라는 변수에 5를 대입하면 a라는 변수에 5라는 값이 저장되어 있다고 하면서 echoing을 한다. 함수를 부르는 경우에도 동일하게 적용이 된다. 만약 함수의 결과를 어떠한 변수에 저장하는 경우에는 그 변수에 값을 저장했다고 echoing을 하게 된다. 그런데 plot과 같은 것도 대부분 사용하는 경우에는 변수에 저장할 값이 없으나 위의 plot의 help의 일부분을 보면 LINE이라는 object의 handles을 벡터로서 리턴해 준다고 했다. 그러므로 다음과 같은 것이 가능해 진다.

```
>>hline=plot(t,y);
```

위와 같이 입력을 하면 plot에 의해 그려지는 그림의 LINE object를 hline이라는 벡터에 저장한다는 얘기가 된다. 그러면 hline에 어떠한 값이 들어갈 수 있는지를 알고 싶은 경우에는 `set`이라는 명령어를 사용해서 가능한 값과 설정을 할 수 있다. `help set`을 해보자.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
SET    Set object properties.
    SET(H,'PropertyName',PropertyValue) sets the value of the
    specified property for the graphics object with handle H.
    H can be a vector of handles, in which case SET sets the
    properties' values for all the objects.

    SET(H,a) where a is a structure whose field names are object
    property names, sets the properties named in each field name
    with the values contained in the structure.

    SET(H,pn,pv) sets the named properties specified in the cell array
    of strings pn to the corresponding values in the cell array pv for
    all objects specified in H.  The cell array pn must be 1-by-N, but
    the cell array pv can be M-by-N where M is equal to length(H) so
    that each object will be updated with a different set of values
    for the list of property names contained in pn.

    SET(H,'PropertyName1',PropertyValue1,'PropertyName2',PropertyValue2,.
    sets multiple property values with a single statement.  Note that it
    is permissible to use property/value string pairs, structures, and
    property/value cell array pairs in the same call to SET.

    A = SET(H, 'PropertyName')
    SET(H,'PropertyName')
    returns or displays the possible values for the specified
    property of the object with handle H.  The returned array is
    a cell array of possible value strings or an empty cell array
    if the property does not have a finite set of possible string
    values.

    A = SET(H)
    SET(H)
    returns or displays all property names and their possible values for
    the object with handle H.  The return value is a structure whose
    field names are the property names of H, and whose values are
    cell arrays of possible property values or empty cell arrays.

    The default value for an object property can be set on any of an
    object's ancestors by setting the PropertyName formed by
    concatenating the string 'Default', the object type, and the
    property name.  For example, to set the default color of text objects
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
to red in the current figure window:

   set(gcf,'DefaultTextColor','red')

Defaults can not be set on a descendant of the object, or on the
object itself - for example, a value for 'DefaultAxesColor' can not
be set on an axes or an axes child, but can be set on a figure or on
the root.

Three strings have special meaning for PropertyValues:
  'default' - use default value (from nearest ancestor)
  'factory' - use factory default value
  'remove'  - remove default value.

See also GET, RESET, DELETE, GCF, GCA, FIGURE, AXES.
```

이 명령어를 보면 좀 복잡해 보이는데 간단히 일단 세가지에 대해서 알면 된다. 우리가 그리고자 하는 그림의 경우에는 이 그림이라는 object가 일단 있어야 된다. 그리고 이 object를 적당히 바꿔야만 되는데 그러기 위해서는 이 object가 가지고 있는 성질을 바꾸면 된다. 그러므로 알아야될 세가지는 다음과 같다.

- 어떤 특성을 바꿀 수 있는가?
- 현재 가지고 있는 특성의 값은 무엇인가 ?
- 바꾸기 위해서는 어떻게 해야 되는가?

이렇게 세가지를 알면 우리가 원하는 것을 할 수 있다. 일단 어떤 특성을 가지고 있는지 알고 싶은 경우에는 `set(handler_name)`를 하면 된다. 그럼 예제였던 경우에는 무엇이 `handler_name`이 될까 ? 좀전에도 얘기를 했지만 `plot`이라는 명령어에 의해 넘어오는 값은 그리고자 하는 그림의 LINE에 관련된 handler가 넘어온다. 그러므로 현재 handler인 hline에 어떠한 값을 변경할 수 있는지 보기위해 `set(hline)`을 실행해 보자.

```
>>set(hline)
        Color
        EraseMode: [ {normal} | background | xor | none ]
        LineStyle: [ {-} | -- | : | -. | none ]
        LineWidth
        Marker: [ + | o | * | . | x | square | diamond | v | ^ | > |
        MarkerSize
        MarkerEdgeColor: [ none | {auto} ] -or- a ColorSpec.
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
        MarkerFaceColor: [ {none} | auto ] -or- a ColorSpec.
        XData
        YData
        ZData

        ButtonDownFcn
        Children
        Clipping: [ {on} | off ]
        CreateFcn
        DeleteFcn
        BusyAction: [ {queue} | cancel ]
        HandleVisibility: [ {on} | callback | off ]
        HitTest: [ {on} | off ]
        Interruptible: [ {on} | off ]
        Parent
        Selected: [ on | off ]
        SelectionHighlight: [ {on} | off ]
        Tag
        UIContextMenu
        UserData
        Visible: [ {on} | off ]

>>
```

아주 많은 것들을 볼 수 있다. 그러면 이것들은 다 바꿀 수 있는 것들인데 잘 보면 현재의 설정값을 볼 수가 있다. 현재 설정되어 있는 값은 '{ }'에 둘러 싸여 있는 값이 설정되어 있는 값이다. 이렇게 보는 것 외에 특정한 한 특성의 값이 어떻게 저장되어 있는가를 보기위해서는 다음과 같이 해 보면 된다.

```
>>set(hline,'linestyle')
```

그러면 현재 설정되어 있는 값을 볼 수 있다. 그 다음 마지막으로 설정된 값을 바꿀려면 다음과 같이 하면 된다.

```
>>set(hline,'linestyle','--')
```

이렇게 명령어를 입력하는 것은 모두 set에 관련된 help 내용에 다 있는 부분이다. 그럼 간단한 예제를 이용해서 바꿔보자. 좀전에 사용했던 step 함수를 그리고자 했던 것으로 부터 정보를 얻어보자.

```
>>hline=plot(t,y)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

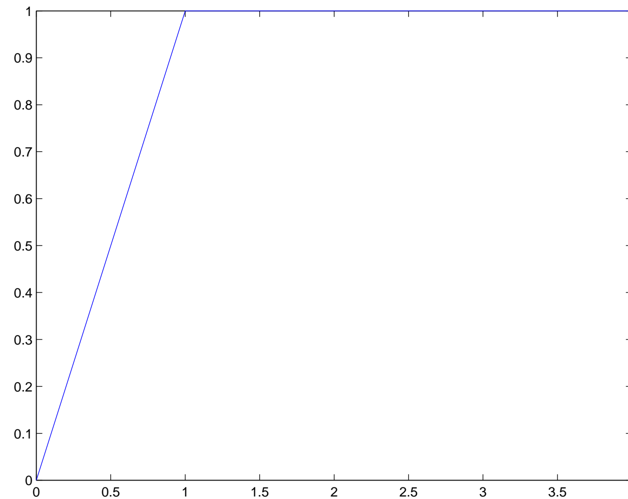

Figure A.27: Advanced plot example: LINE object handler 1

그러면 예상했던 것과 동일한 그림 A.5.3을 얻을 수 있고 그와 동시에 hline이라는 변수에 LINE과 관련된 handler를 치환해 두었다. 그 다음 이 handler의 값을 보기 위해서는 set(hline)라는 명령어를 입력하면 된다. 그 결과는 위에 있는 set을 이용한 경우와 동일하기 때문에 생략한다. 그 다음 handler의 값을 하나 바꿔보자. 일단 저장되어 있는 값을 확인하기 위해서 set(handler_name,'property_name')을 해보자.

```
>>set(hline,'linestyle')
[ {-} | -- | : | -. | none ]
>>
```

위와 같이 현재 linestyle은 '-'로 되어 있다. 이 값을 '-.'로 바꿔보자.

```
>>set(hline,'linestyle','-.')
```

이 값을 입력하면 무슨 변화가 생길까? 그림을 보자. 그림 A.5.3와 같이 라인의 형식이 바뀌었다. 원래 이렇게 하기 위해서는 기본적으로 plot을 할 때 바꿔줘야만 되는데 굳이 이미 만들어진 그림이라면 이렇게 해서 바꿀 수 있다. 자 여기까지 배운 것을 이용해서 plotyy에 의해 만들어진 그래프의 형태를 바꿔보자.

이번에는 plotyy에 대한 help 내용을 참고해보자.

```
[AX,H1,H2] = PLOTYY(...) returns the handles of the two axes created
    AX and the handles of the graphics objects from each plot in H1
    and H2. AX(1) is the left axes and AX(2) is the right axes.
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

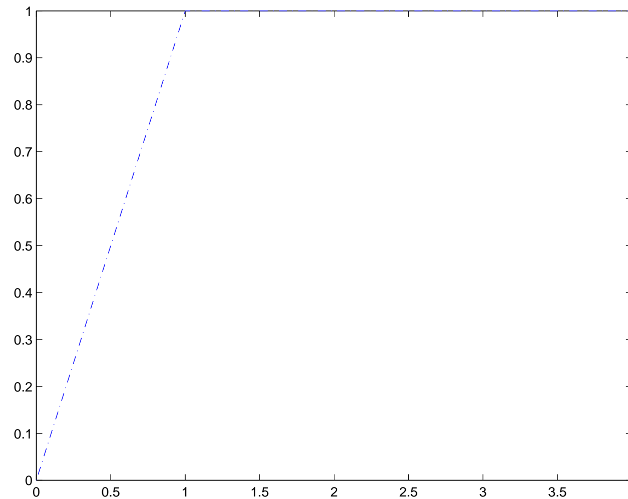

Figure A.28: Advanced plot example: LINE object handler 2

이 부분은 plot의 명령어의 경우와 유사하지만 돌려주는 것이 두 그래프의 handler를 각각 H1과 H2라는 변수로 돌려주고 두 축에 대한 것을 AX라는 변수에 돌려준다고 되어 있다. 우리가 지금 하고자 하는 것은 각각의 y 축에 각각의 라벨을 붙이는 것이다. 그러면 일단 각각의 축을 선정해 주어야된다. axes라는 함수의 help 파일 내용을 보면 다음과 같은 부분이 있다.

```
AXES, by itself, creates the default full-window axis and returns
a handle to it.

AXES(H) makes the axis with handle H current.
```

즉, axes라는 명령어 자체로는 현재 윈도우에 축을 만들고 그 그림에 handle을 돌려주는 것인데 마지막 부분은 보면 H라는 현재 handle의 축을 만든다고 되어 있다. 그러므로 그래프에서 임의의 y축을 선정할 수 있다. 자 그럼 바꿔보자.

```
>>[haxes, hline1, hline2]=plotyy(t,y,t,y,'plot','stairs');
>>axes(haxes(1))
>>ylabel{Simple Plot}
>>axes(haxes(2))
>>ylabel{Stairs Plot}
```

위의 명령어에 의하면 각각의 축을 선정하고 난 후 그 축에 이름을 붙였다. 결과 그림은 다음과 같다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

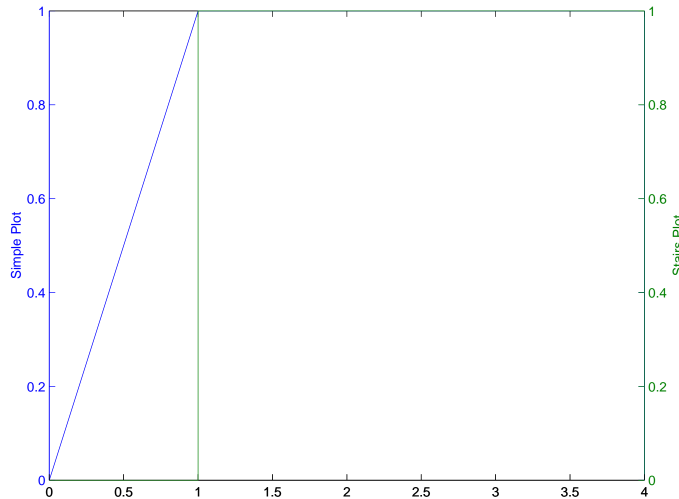

Figure A.29: Advanced plot example: plotyy with ylabel

이것들말고 3차원 그림을 그리는 것이 남아있는데 이부분은 현재 작성하지 않겠다. 아마도 3차원 그림을 그리는 수치해석 문제는 다루지 않을거 같기 때문이다. 그리고 좀더 세세하게 그림의 각 부분을 수정하는 경우도 있는데 이 부분도 여기서 설명하지는 않겠다. 혹 나중에 사용할 필요가 있는 경우는 지속적으로 추가를 해 나가겠다.

A.6 예제

이번장에서는 몇가지 예제를 만들어보면서 좀더 matlab에 친숙해지는 기회를 만들어 보자. 예제에서는 여러가지를 다룰 예정이면 계속해서 업데이트를 해 나갈 것이다.

A.6.1 Linear Equation

이제부터 설명한 부분에서 사용할 행렬은 다음과 같다. 다음에 있는 행렬은 matlab 설명서에 있는 것을 이용했다.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \tag{A.2}$$

두 행렬의 합을 계산하라. 즉, $A + B$는 얼마인가 ? 이 문제를 풀어야 된다면 여러분들이 어떻게 해결할 것인가? 자 그럼 지금까지 배운것을 바탕으로 한번 일단 생각을 해 보자. 두 행렬의 합을 계산하라고 하는 것은 행렬의 각각의 요소를 하나씩 더해 나가면 된다. 즉, 다음과 같이 프로그램을 만들 수 있을 것이다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
A=[1 1 1; 1 2 3; 1 3 6];
B=[8 1 5; 3 5 7; 4 9 2];
for i=1:3
   for j=1:3
      C(i,j)=A(i,j)+B(i,j);
   end
end
```

위의 장에서 설명을 했듯이 행렬의 각 요소를 나타내기 위해서는 요소에 해당하는 인덱스를 지정해 주면 된다. 그리고 반복적인 계산을 수행하기 위해서는 for 문을 이용해서 계산하면 된다. 한번 생각을 해보자. 위에 작성한 프로그램은 지금 3행 3열 행렬만을 다룰 수 있는 프로그램이다. 이것을 그러면 주어진 A와 B의 형태에 상관없이 작동이 될려면 어떻게 작성을 해야할까 ? 여러분이 명심해야될 사항은 여러분 말고도 누군가 여러분과 똑같은 생각을 이미 했다는 사실이다. 그러므로 항상 컴퓨터를 사용할 때에는 여러분이 아직 못찾거나 사용하지 못하고 있을 뿐이지 필요한 기능이 없을거라고 생각하면 된다.(이 강의록의 저자는 항상 그렇게 생각한다. 적어도 큰회사에서 만들어진 프로그램이라면) 저번에 이미 했었다. 뭘 이용하면 행렬의 크기에 상관없이 행렬합을 계산하는 루틴을 만들 수 있을까. length와 size라는 함수를 기억하는 가? 이 두함수는 약간 다르다. 단순히 행렬의 크기만을 계산한다면 length라는 함수가 적당할 것이고 행렬의 정확한 크기를 알고 싶다면 size 함수를 사용하는 것이 좋을 것이다. 위의 계산 루틴을 수정해서 다음과 같이 만들어 봐라.

```
A=[1 1 1; 1 2 3; 1 3 6];
B=[8 1 5; 3 5 7; 4 9 2];
for i=1:length(A)
   for j=1:length(A)
      C(i,j)=A(i,j)+B(i,j);
   end
end
```

이 프로그램은 행렬의 크기에 상관없이 행렬의 합을 계산할 수 있다. 그렇지만 좀더 생각을 해 보자. 만약 행렬이 square matrix가 아니고 rectangular matrix 라면 어떻게 할 것인가 ? 즉, 이번에는 size 함수를 사용해야만 한다는 결론을 얻을 수 있다. size 함수는 두개의 출력값을 주는데 하나는 행의 값이고 하나는 열의 값이다. 그럼 다시 한번 고쳐보자.

```
A=[1 1 1; 1 2 3; 1 3 6];
B=[8 1 5; 3 5 7; 4 9 2];
[m n]=size(A);
for i=1:m
   for j=1:n
      C(i,j)=A(i,j)+B(i,j);
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      end
   end
```

자 이제 어느정도 완성을 했다. 두 행렬이 주어진다면 언제든지 계산을 할 수 있는 준비가 되었다. 행렬 값만 중어진다면 그 행렬의 크기를 계산하고 그 값을 이용해서 자동으로 행렬 크기에 대한 인덱스를 만들어 계산을 할 수 있다. 그런데 이렇게 하면 끝일까? 한번 더 생각을 해보자. 만약 두 행렬의 크기가 다르다면 그러면 어떻게 할 것인가. 여러분은 당연히 그럴 일이 없다고 할지 모르겠지만 그런일이 생긴다면 어떻게 할 것인가. 그것에 대한 대비책을 만들어 보자.

```
A=[1 1 1; 1 2 3; 1 3 6];
B=[8 1 5; 3 5 7; 4 9 2];
[m1 n1]=size(A);
[m2 n2]=size(B);
if(~(m1==m2) | ~(n1==n2))
   error('똑바로 넣으란 말야');
end
for i=1:m1
   for j=1:n1
      C(i,j)=A(i,j)+B(i,j);
   end
end
```

적어도 이렇게 까지 만들어 두면 문제가 발생하지는 않는다. 그럼 마지막으로 위에서 만든것을 늘 그냥 행렬의 합을 계산할 때마다 카피해서 넣어야만 된다. 이것 또한 얼마나 불합리한 일인가? 이런 경우에 사용하라고 함수라는 것이 있는 것이다. 함수를 만들어 두면 예를 들어 간단히 matrix_add(A,B) 라고 하면 두 행렬의 합을 계산해 줄 수 있다. 이제 그것을 만들어 보자.

```
function plus_out=matrix_add(a, b)

[m1 n1]=size(a);
[m2 n2]=size(b);
if(~(m1==m2) | ~(n1==n2))
   error('똑바로 넣으란 말야');
end
for i=1:m1
   for j=1:n1
      plus_out(i,j)=a(i,j)+b(i,j);
   end
end
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

이렇게 함수를 만들고 난 후 matrix_add.m으로 저장을 하고 간단히 메인 문에서는 다음과 같이 부르면 된다.

```
>>A=[1 1 1; 1 2 3; 1 3 6];
>>B=[8 1 5; 3 5 7; 4 9 2];
>>C=matrix_add(A,B);
```

처음에 시작했던 프로그램보다 상당히 많이 깔끔하고 알아보기 쉽게 만들어지지 않았는가 ? 이렇게 프로그램을 짜야 나중에 다시 보거나 아니면 에러를 찾을때도 matrix_add 함수를 한번만 보면 되지 아니면 계속 이와 같은 루틴을 계속 찾아 버그를 찾아야 되기 때문에 상당히 피곤한 일이된다.

그렇지만 어쩌나. 난 지금까지 여러분을 괜히 고생을 시켰다. matlab 이라는 이름을 잘 보자. 뭔가 matrix와 관련이 있어보이지 않는가 ? 그렇다. 이 matlab은 다른 것은 몰라고 행렬과 관련된 계산은 정말 잘 해준다. 즉, 위에서 했던 예제는 정말 예를 만들기 위해 만든 것이고 사실 그런건 필요없다. matlab에서는 이미 얘기를 했지만 변수가 단순한 값을 가지고 있는 변수인지 아니면 행렬인지 선언을 해주지 않는다고 했다. 그러므로 계산할 때에도 동일하다.

```
>>A=[1 1 1; 1 2 3; 1 3 6];
>>B=[8 1 5; 3 5 7; 4 9 2];
>>A+B
```

이렇게 해주면 그냥 끝이다. 만약 결과 값을 다른 변수로 받고 싶으면 단순히 그 관계식만 표현을 하면 된다. 뺄셈도 상관없고 곱셈도 또한 * 기호를 써서 나타내면 된다. 그렇다면 나눗셈은 ? 행렬에 나눗셈이 있을까 ? 행렬은 나눗셈이라고 하지 않고 역행렬을 계산해서 넘겨줘야 된다.

$$Ax = B \tag{A.3}$$
$$x = A^{-1}B \tag{A.4}$$

아무대나 그냥 역행렬을 곱하면 안된다. 정확히 순서를 지켜서 곱해줘야된다. 이런 내용은 이미 여러분이 배웠을꺼라 생각을 하겠다. 이걸 matlab에서 구현할때는 크게 세가지로 할 수 있다.

```
>>A^-1 * B
>>inv(A) * B
>>A \ B
```

첫번째꺼는 A라는 행렬에 -1 승을 한다는 의미로 사용한 것이고 두번째꺼는 inv라는 inverse를 구해주는 함수를 이용한 것이다. 그리고 마지막으로 세번째꺼가 있는데 이것은 좀 설명을 해

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

야겠다. matlab에서는 다음과 같은 표기법을 사용한다.

$$
\begin{aligned}
Ax &= B \\
x &= A\backslash B \\
xA &= B \\
x &= B/A
\end{aligned}
$$

위의 표기법을 잘 보면 어떻게 되던지 나누는 것을 밑으로 쓴다고 보면 된다.

이번 예제의 제목은 Linear equation이다. 이 문제의 해결방법은 너무도 간단히 해결될 수 있다. 어떻게 하던지 간에 위에서 봤듯이 x라는 벡터나 행렬의 값을 계산을 하면 된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

Appendix B

Using Fortran

포트란은 이미 오래전부터 수치해석의 툴로서 사용이 되어 왔다. 그러나 72column의 제약이나 언어를 사용하는데 있어 여러가지 제약이 있기 때문에 사용하기에 어려운 점이 있다. 그래픽과 관련된 함수들이 많지 않고 Visual한 측면은 부족하지만 수치해석만을 고려한다면 상당히 빠르고 뛰어난 언어이다. 현재까지 많은 사람들이 사용해 왔고 또한 많은 함수나 서브루틴 등이 포트란으로 짜여져 있기 때문에 배워야할 필요가 있는 언어이다. 또한 사용하다보면 여러가지 면에서 편한 점들도 있다.

B.1 설치 및 사용법

최신 버전의 Visual fortran을 사용하것 보다 여러분이 주위에서 쉽게 구할 수 있는 포트란 프로그램을 사용하는 것이 적당하다. 또한 Unix나 linux서버에서는 기본적으로 포트란을 사용할 수 있다. 각 경우에 맞춰 설명하겠다.

B.1.1 MS Window에서 작동하는 포트란

윈도우에서 작동하는 포트란의 경우 설치는 기본적인 방법을 따라 설치를 한다. 그 다음 프로그램을 사용하기 위해서 윈도우를 이용할 수도 있지만 굳이 프로그램을 띄우기 위해서 높은 사양의 컴퓨터를 사용하는 것보다 단순히 도스창에서도 모든 것을 할 수 있기 때문에 도스창에서 사용하는 명령법에 대해 배운다.

간단한 프로그램인 sample.for 프로그램을 다음과 같이 작성을 한다.

```
C        1         2         3         4         5         6
C23456789012345678901234567890123456789012345678901234567890123456789
```

```
C
C Sample file to show basic fortran program
C
      program sample

      integer i, j, sum

      i=1
      j=2

      sum=i+j

      write(*,10) i, j, sum
   10 format(3i)
      stop
      end
```

위에있는 첫번째 두줄은 현재의 컬럼을 보여주기 위해 작성한 것이다. 첫번째 컬럼에 C를 쓰고 사용하는 줄은 그 라인이 comment라는 것을 의미한다. 실제 컴파일시 처리되지 않는 부분이다.

실제 프로그램은 7번째 컬럼부터 시작하며 총 72컬럼까지 프로그램을 작성할 수 있다. 이러한 규정은 사실 요즘의 최신 컴파일와 fortran99 규정에는 어느정도 완화된 편이나 프로그램을 작성할때에는 기본적인 문법에 맞춰 작성하는 것이 가장 좋다. `program` 키워드는 프로그램이 시작되는 것을 의미하며 변수들이 사용되기 위해서는 사용하기 전에 변수의 형에 대한 선언이 필요하다. 위에있는 예제 프로그램에서는 세개의 정수형변수가 선언되었다. 자세한 내용은 다음에 다룰 것이다.

위와 같이 프로그램을 작성을 한 후 소스코드를 실행코드로 만들기 위해서는 컴파일러를 사용해야되는데 윈도우에서 작동되는 프로그램에서는 `fl32.exe`를 사용한다.

```
fl32 sample.for
```

위의 명령어는 sample.for 파일을 컴파일하고 그 다음 링크를 해서 `sample.exe` 실행파일을 생성한다. 여러분들이 혼동해서 사용하는 경우가 있는데 컴파일과 링크는 다른 의미이다. 혹자들은 이것을 같은 것으로 사용하는 경우가 있는데 컴파일을 하는 것은 소스코드로 부터 목적코드 즉, object file을 얻는 것이고 이 object code를 다른 라이브러리와 함께 같이 링크해서 얻는 코드가 실행 코드이다. 다시 한번 생각을 해 보자.

$$\text{source code} \xrightarrow{compile} \text{object code} \xrightarrow{link} \text{executive code}$$

현재 사용하고 있는 언어가 포트란이므로 이 경우에는 source code는 확장자가 .for로 되어 있고 목적코드 .obj 실행코드는 .exe로 만들어진다. 실행코드를 만들 때에는 언어의 문법적 오류

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

만을 체크한다. 그냥 fl32 명령어를 실행하는 경우에는 실행코드를 만들기 위해 컴파일과 링크를 순차적으로 실행을 한다. 작은 프로그램이나 간단한 프로그램을 만드는 경우에는 큰 차이가 없지만 좀 더 큰 프로그램을 만드는 경우에는 일단 프로그램을 대강 작성하고 난 후에는 컴파일만 해서 프로그램의 오류를 검사해야만 한다. 이렇게 하기 위해서는 /compile_only 라는 옵션을 사용하는데 간단히 /c 를 이용해도 된다.

```
fl32 sample.for /c
```

이것으로 일단 MS 윈도우에서 포트란 소스 코드를 이용해서 실행파일을 만드는 방법에 대해서 논의하였다. 이것 외에 나중에 라이브러리를 생성하는 방법등에 대해 다룰 것이다. 다음에는 Unix 머신에서 컴파일하는 방법에 대해서 논의를 해 보자.

B.1.2 Unix 머신에서 작동하는 포트란

유닉스머신이나 리눅스 머신에서 포트란을 사용하기 위해서는 일단 시스템 관리자에서 포트란을 설치해 달라고 요청을 해야된다. 요즘에 관심의 촛점이되고 있는 리눅스 머신의 경우 기본적으로 포트란 컴파일러가 설치가 되어 있다. 좀 전에 작성한 `sample.for` 파일을 리눅스 머신에서 컴파일하는 방법에 대한 생각을 해보자. 리눅스 머신에서 사용하는 명령어는 대부분 `f77`이나 `g77`이다. `g77`의 경우를 예를 들어 설명하겠다.

```
g77 sample.for
```

위와 같은 명령어에 의해 생성되는 파일은 윈도우의 경우와는 약간 다르다. 위의 경우에는 실행 파일의 이름을 정해주지 않았기 때문에 `a.out`이라는 형태의 실행파일이 만들어졌다. 그러므로 프로그램을 실행하기 위해서는

```
./a.out
```

이라고 해야만 실행이 된다. 실행 파일의 이름을 결정하기 위해서는 -o 옵션을 사용해야 된다.

```
g77 -o sample sample.for
```

또한 유닉스나 리눅스에서는 도스와는 달리 실행파일의 확장자 같은 것은 필요가 없기 때문에 파일의 실행 권한만 있으면 된다. 그러므로 위에서 컴파일한 파일을 실행하기 위해서는

```
./sample
```

이라고 하면 된다. 또한 단순히 목적코드만을 만드는 경우에는 -c 옵션을 사용하면 된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
g77 -c sample.for
```

이 경우에는 윈도우 시스템과 달리 sample.o 라는 파일을 만들어 준다. 윈도우 시스템에서는 sample.obj 가 만들어지는 것과는 약간 다르다.

마지막으로 다른 언어와 달리 포트란은 대소문자를 구분하지 않는다. 윈도우 시스템인 경우야 당연이 시스템을 사용하는 경우 대소문자를 구분하지 않지만 유닉스 시스템인 경우는 다르게 대소문자를 구분한다. 예를 들어 a.exe와 A.exe 파일이 있는 경우 윈도우 시스템은 같은 것으로 인식을 한다. 또한 명령어를 사용하는데 있어서도 디렉토리를 바꾸는데 사용하는 명령어인 'cd'나 'CD'를 구분하지 않는다. 즉, 같은 명령어로 인식한다는 것이다. 그러나 유닉스 시스템에서는 대소문자를 구분해서 위에서 예를 들은 a.exe와 A.exe라는 파일은 다른 파일이며 명령어인 'cd'는 있지만 'CD'라는 명령어는 없는 것으로 나온다. 그렇지만 두 시스템 모두 포트란에서 사용하는 변수나 함수는 모두 동일한 것으로 인식을 한다.

```
C        1         2         3         4         5         6
C23456789012345678901234567890123456789012345678901234567890123456789
C
C Sample file to show basic fortran program
C
      program sample

      integer a, B

      a=1
      B=2

      write(*,10) A, b
   10 format(2i)
      stop
      end
```

위의 예제를 컴파일 하는 경우 어떠한 에러도 발생하지 않는다. 그러나 대부분의 다른 언어의 경우에는 이 두개를 구분해서 사용하므로 포트란을 사용할때 주의를 해야된다. Matlab에서도 a라고 지정한 변수와 A라고 지정한 변수는 다른 것으로 알고 있다.

B.1.3 Summary

지금까지 내용을 정리해 보면 다음과 같다.

- 72칼럼의 제약, 6번째 컬럼은 연결을 의미, C나 *를 이용한 comment 처리

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- 프로그램은 실제 7번째 컬럼에서 시작됨
- 컴파일만 하기위해서는 윈도우에서는 /c 옵션사용 유닉스에서는 -c 사용
- 컴파일과 링크의 차이점
- 대소문자 구분의 특징

B.2 데이터와 입출력

언어를 배우는데 있어서 기본적으로 알아야될것은 변수의 선언방법, 제어문 작성방법, 부프로그램 작성방법에 대해서 알면 기본적인 것은 다 배운 것이라고 볼 수 있다. 포트란 프로그램을 작성하기 위해서는 포트란 언어의 문법에 맞는 특징에 대해 배우면 된다.

B.2.1 기본적 구성

포트란 언어는 다른 언어들과는 달리 지켜야 될 문법이 몇가지 있다. 이전 장에서 이미 약간 설명을 하였지만 정리를 하면 다음과 같다.

- 첫번째 컬럼이 C나 *로 시작하는 경우는 comment 문이다.
- 프로그램의 내용은 7번째 컬럼부터 시작한다.
- 프로그램은 72컬럼내에서 작성을 해야된다.
- 프로그램이 길어지는 경우에는 6번째 컬럼에 0을 제외한 다른 문자를 넣어 연결된 문장이라는 의미를 갖게한다.
- 1번째 부터 5번째까지는 문번호를 입력한다.

B.2.2 기본적 데이터 타입

기본적인 데이터 타입은 integer, real, character 등이 있다. 포트란 프로그램을 작성하는 경우 변수에 대한 형선언이 없는 경우에는 i-n로 시작하는 변수는 integer 변수로 여겨지고 나머지 변수는 real 변수로 인식한다. 그리고 정확도가 요구되는 경우에는 double precision이라는 형태의 변수를 사용한다. double이라고 선언이 된 변수는 real변수에 비해 정확도가 증가된다. 그러므로 변수의 값을 정확히 저장하는 경우에는 double변수를 사용해야된다. 그러나 변수가 double로 선언이 되는 경우에는 메모리를 더 많이 차지하게 된다. 대부분의 경우 수치해석을 하는 경우에는 double형 변수를 많이 사용하게 된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

그리고 이러한 기본적인 형 선언과 다른 형태의 변수를 사용하고자 하는 경우에는 꼭 선언을 해 주어야 된다. 다음은 예제이다.

```
C        1         2         3         4         5         6
C234567890123456789012345678901234567890123456789012345678901234567890
C
C Sample file to show basic fortran program
C
      program sample

      double precision dsum

      i=1
      j=2
      sum=10.0
      dsum=10.1+20.2
     &    +30.3

      write(*,10) i, j
   10 format(2i)
      stop
      end
```

위의 예제에서는 변수 i, j, sum에 대해 변수 선언부분이 없다. 그 이유는 좀전에 설명한 것과 같이 기본적으로 i-n(i,j,k,l,m,n)로 시작하는 변수는 정수형변수가 되기 때문이다. 그러므로 i,j는 정수형 변수이며, sum은 실수형 변수가 된다. 마지막으로 dsum의 경우에는 double 형 변수이다. 실행 파일을 만들어 보면 dsum을 real로 선언한 경우보다 실행파일의 크기가 증가되는 것을 발견하게 된다. 그러한 이유는 dsum이라는 변수를 double로 저장하기 위해서는 더 많은 메모리를 사용해야되기 때문이다. 예전에 도스만을 사용하는 경우에는 640kb의 메모리 제약으로 변수를 사용하는 경우 메모리에 대한 고려를 상당히 했어야만 되지만 현재는 개인적인 수치해석 프로그램을 작성하는 경우에는 메모리 문제를 심각하게 고려할 필요는 없다고 본다. 물론 그렇다고 메모리를 낭비할 필요는 없다. 이러한 메모리까지 고려하는 경우는 여러분들이 상당한 수준의 프로그램을 짜는 경우에 고려해야될 사항이다.

그리고 형 선언과 관련된 선언문으로 parameter와 implicit 문이 있다. parameter의 경우에는 상수를 선언하는 경우에 사용이 된다. C에서 #define과 비슷한 것이다. 변수를 크게 두개로 나누는 경우에는 상수와 변수가 있다. 이말은 어떻게 보면 이상하지만 이렇게 표현할 수 밖에 없다. 상수라고 하는 것은 프로그램을 작성하는 경우 계속해서 변하지 않는 값을 가지고 있는 기억 장소를 의미하고 변수라고하는 것은 변할 수 있는 값을 저장하는 경우 사용한다. 그러므로 parameter로 선언된 변수는 프로그램이 실행되는 동안 변하지 않는 변수를 지정하기 위해서 사용된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

implicit문은 변수의 생성시 위에서 얘기했듯이 규칙에 따라 i-n로 시작하는 경우 특별한 변수 선언이 없는 경우 real로 선언이 되는데 만약 i로 시작하는 real변수를 만들기 위해서는 여러분은 형 선언을 계속해서 사용해야 된다. 그렇지만 이러한 변수를 선언하는 작업은 상당히 귀찮은 작업이다. 이경우 implicit문을 사용하면 i로 시작되는 모든 변수를 real로 선언할 수도 있고 또한 double로 선언할 수도 있다.

```
C        1         2         3         4         5         6
C23456789012345678901234567890123456789012345678901234567890123456789
C
C Sample file to show basic fortran program
C
      program sample

      implicit double precision(a-h,o-z)
      parameter(n=10)

      i=1
      j=2
      sum=10.0
      dsum=10.1+20.2
     &    +30.3

      write(*,10) i, j
   10 format(2i)
      stop
      end
```

위와 같은 프로그램에서는 n은 10이라는 값을 가지는 상수로서 선언이 되었고 sum과 dsum은 모두 double로 선언이 된다.

여러가지 변수의 형태가 다른 변수를 혼용해서 사용하는 경우 값은 계산된 형태에 따라 다르게 나타난다. 또한 정수형과 실수형을 혼합해서 사용하는 경우는 나머지 값이 없어진 형태로 값이 전달된다는 것을 주의하기 바란다. 1/2의 정수형 변수 값은 0이고 또한 이 식의 값을 실수형 변수로 대입을 해도 이미 계산 결과가 정수형에서 계산이 되었기 때문에 0이 값이 대입된다. 그러나 1/2.은 정수형인 변수로 치환되는 경우에는 0이나 실수형으로 치환이 되는 경우에는 0.5이다. 두번째에서는 '2.' 이 사용된 것을 잘 보기 바란다. 실수형 계산을 정수형으로 바꾸는 것은 상당히 안좋은 프로그램 방법이다. 이런 경우에는 실수형 값을 적당한 함수를 이용해서 몫과 나머지로 바꿔서 사용해야 된다. 이 부분은 나중에 기회가 되면 설명을 하겠다.

```
      program test
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
i=1
a=i+1
i=a/i
j=i/a
b=1/2.

write(*,*) a, i, j, b

stop
end
```

위의 예제에서 a값은 2.으로 출력이된다.(주의 : 2가 아니라 2. 즉, 실수라는 것) 그러나 i 값의 경우는 2로 출력이 된다. 그러면 j값은 어떠한 값이 출력이 될까 ? 그리고 마지막으로 b는 어떤 값을 출력이 될지 잘 생각해 보고 직접해보기 바란다.

좋은 프로그램밍 방법은 사용하고자 하는 변수를 전부 다 선언을 하고 사용하는 것이 좋다. 왜냐면 프로그램의 길이가 길어지는 경우 현재 자신 프로그램 내에서 어떠한 변수를 사용하고 있는지 알기가 힘들어진다. 그러므로 사용하고자 하는 변수를 전부 다 형선언문에 넣어주는 것은 좋은 프로그래밍 방법이다.

마지막으로 변수에 대한 선언에서 조심해야 될 점은 간혹 실수로 선언한 후에 그 값을 정수와 비교하는 경우가 있다. 그러한 경우는 올바른 값을 얻을 수 없을 것이다. 이 부분은 조금 뒤 제어문에 대한 설명을 하는 부분에서 좀더 다루겠다.

이제 포트란 언어에 대한 설명을 좀더 해야겠다. 처음에 설명을 했던 matlab과 같이 이제부터는 프로그램이 나오는 것에 따라서 조금씩 설명을 해 나가겠다. 위에 예제에서 보면 write라는 명령을 사용했는데 정확히 write(*,*)라고 하였다. 정확히 write를 사용하기 위해서는 write(unit, format_no) 형태로 사용해야 된다. unit라고 하는 것은 write에 의해서는 어떤 결과가 출력이 되는데 그 결과를 출력할 unit를 정하라는 것이다. 이게 도대체 무슨 말인가 ? 결과를 출력을 하기 위해서는 출력물을 쓸 장치가 필요하다는 것이다. 일반적으로 사용하는 장치는 화면과 키보드이다. 즉, 입력을 받을때는 키보드로 받고 출력물을 쓸때는 화면에 출력을 한다. 대부분의 경우 5번과 6번을 키보드와 스크린으로 사용하고 나머지는 사용자가 정한 것에 따라 사용하게 된다. 그런데 만약 이렇게 귀찮게 unit의 숫자를 정하지 않고 그냥 키보드와 화면을 정하고 싶을때는 어떻게 해야할까 ? 그런 경우에는 '*' 기호를 사용하면 자동으로 할당을 해 준다. 그리고 두번째 format_no는 format문의 문 번호를 의미한다. 만약 지금 출력하거나 입력할 변수의 형태를 정해주기 위해서는 format문에 이것에 대한 정보를 입력해야 된다. 그렇지만 만약 그냥 주어진대로 출력을 하고 입력을 하기 위해서는 '*'를 이용한다. 이 *를 이용하는 경우에는 특별한 형태를 정해주지 않는다는 것을 의미한다. 자세한 내용은 다음에 설명을 하겠지만 위의 예제는 변수 4개를 출력하는데 특별한 형태를 정하지 않고 출력하는 것이다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

그 다음 프로그램의 실행이 종료된다는 것을 나타내기 위해서 stop이라는 키워드를 사용했고 마지막으로 end 키워드로 프로그램의 끝났다는 것을 나타낸다.

데이터의 형태를 정해주는데 integer나 real, double precision을 사용하다는 것은 이미 했지만 이 것말고 문자열과 관련되 character나 complex, logical 등이 있다. 여러분들은 아마도 character 정도를 사용하게 될 것이다. character를 사용하는 예제를 만들어 보자.

```
      program test

      character i, a
      character*8 j, b

      i='I'
      j='I am Tom'

      write(*,*) i, j
      write(*,10) i, j
   10 format(a1, 1x, a8)
      write(*,20) i, j
   20 format(1x, a1, 5x, a8)

      a='You'
      b='You are Jane'

      write(*,*) a, b

      stop
      end
```

이 예제를 한번 잘 보자. 처음 부분에 변수를 선언하는데 character문을 사용했는데 두번째 character문에는 좀 이상한 것이 붙어 있다. 기본적으로 그냥 character를 선언하고 사용하면 그 변수는 문자 하나가 된다. 문자열은 정확히 string이고 문자 하나는 character이다. 그러므로 변수 i나 a는 글자 하나만을 가질 수 있는 변수이다. 그 다음 글자 하나가 아니라 여러개인 경우에는 character*갯수 형식으로 사용한다. 위에서 j와 b는 8개까지의 글자를 가질 수 있는 변수이다. 문자 변수에 값을 대입하기 위해서는 ' ' '나 ' " '를 사용해서 입력하고자 하는 내용을 둘러 싸야만 된다. 그리고 변수의 값에는 빈칸도 하나의 문자로 입력되는 것을 기억하자.

자 그럼 i와 j 변수에는 위에서 선언한 값과 동일한 크기의 변수가 잘 대입이 되었다. 'I'는 1크기이고 'I am Tom'은 빈칸을 포함해서 8개이다. 처음 출력부분은 i와 j 변수를 출력하기 위해서 포멧문을 사용하지 않았다. 출력되는 결과는 'II am Tom'이 될것이다. 두 변수 사이에 뛰어 쓰기도 할 수 없으므로 당연한 것이다. 다음 두번째 출력 부분은 포멧문을 정해 두었다. 10번 문번호를 가지는 포멧문을 보면 일단 문자열 변수를 출력하기 위해서는 'a'라는 지시자

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

를 사용한다. 그리고 그 a자 뒤에 붙이는 숫자는 크기를 의미한다. 즉, a1라고 하는 것은 문자열 1자리 출력, a8이라고 하면 8자리 글자 출력이라는 의미다. 그런데 이렇게 출력을 정해주는 지시자 앞에 숫자를 붙이는 경우가 있는데 이런 경우는 숫자 다음의 지시자를 반복적으로 몇번 사용할지를 정해주는 것이다. 만약에 3a2라고 되어 있으면 a2라는 출력포멧 형식을 세번 사용하겠다는 것이다. 자세한 것은 뒤에 설명을 하자. 일단 처음에 i변수의 값을 한자리로 출력을 하고 그 다음 x라는 지시자가 있는데 이것은 칸을 띄우라는 지시자다. 일단 지금은 한칸을 띄운다. 그 다음 변수를 8자리로 찍으라고 했다. 그러니 결과는 'I I am Tom'이 나올것이라고 예상할 수 있다.

그 다음 다른 예제를 보면 a와 b라는 변수도 위에서 사용한 것과 동일하게 크기를 가지는 변수로 선언을 했지만 그 선언된 변수 값보다 더 큰 값을 대입을 했다. 그 결과는 그래봐야 소용이 없다는 거다. 한글자 변수로 선언을 햇으면 한글자만 입력을 해야지 괜히 세글자를 넣어봐야 아무런 영향을 미치지 못한다. 출력 결과는 'YYou are '로 나온다. 정확히 한글자와 8글자로 구성이 된 결과이다.

B.2.3 입력과 출력에 관해

이미 이 부분은 많이 설명을 했지만 이번에 정리를 했으면 한다. 사실 여러분은 open문도 사용할 수 있어야만 한다. 자 위에서 설명한 것과 추가로 설명할 것을 정리해 보자.

입력과 출력은 크게 read와 write를 사용한다. 물론 이것외에 print도 사용할 수 있다. 일단 read와 write에 대해 알아보자.

```
read(unit, format)
write(unit, format)
```

둘다 입력이나 출력에 사용할 unit을 정해주고 또한 출력할 포멧을 정해준다. 기본값을 사용하는 경우에는 그냥 '*'를 사용하면 된다. 여기서 unit에 대해서는 이미 얘기를 했었다. 그렇지만 더 자세히 얘기를 해 보자. 지금 현재는 사용하는 unit의 숫자가 5,6번이다. 이 번호가 각각 입력과 출력을 담당하고 있다. 그러면 만약 여러분이 어떠 파일을 정해서 이 파일을 읽어서 변수 값을 대입하고 그리고 출력할 결과는 여러분이 정한 파일이 써야되는 경우에는 어떻게 해야 될까 ? 그 경우에 대비해서 open 문이 있다. open문의 형식은 다음과 같다.

```
OPEN ([UNIT=]io-unit [, FILE=name] [, ERR=label] [, IOSTAT=i-var], sl
```

복잡해 보이는데 나중에 예제는 보면 좀 더 편안히 볼 수 있을 것이다. 일단 처음에 Unit 숫자를 정해준다. 그 다음 파일명을 결정을 하고 일단 이렇게만 하자. 더 자세한 것은 얘기하지 말고 그럼 예제를 한번 만들어 보자.

```
        program test
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
open(unit=9,file='test.in')
open(unit=10,file='test.out')

read(9,*) a
write(10,*) a

stop
end
```

아주 간단히 만든 예제이다. 잘 보면 아마도 9번 unit에서 파일을 test.in이라는 파일을 열고 10번 unit에서 test.out이라는 파일을 여는 것으로 보인다. 아직까지는 이 파일들이 어떤것을 입력으로 사용할지 아니면 출력 값을 지정하는 것으로 사용할지 정하지 않았다. 위와 같이 프로그램을 작성한 후 test1.for라고 저장을 한 후 실행 파일을 만들어서 실행해 보면 다음과 같은 결과를 얻는다.

```
F:\work>test1
forrtl: severe (24): end-of-file during read, unit 9, file test.in
Image              PC        Routine         Line        Source
test1.exe          0040A0C2  Unknown         Unknown     Unknown
test1.exe          00409EC3  Unknown         Unknown     Unknown
test1.exe          00409084  Unknown         Unknown     Unknown
test1.exe          004092FE  Unknown         Unknown     Unknown
test1.exe          00403B22  Unknown         Unknown     Unknown
test1.exe          004010DB  Unknown         Unknown     Unknown
test1.exe          0042E6B9  Unknown         Unknown     Unknown
test1.exe          004258B4  Unknown         Unknown     Unknown
KERNEL32.dll       77E57903  Unknown         Unknown     Unknown
F:\work>
```

실행을 하고 난 후 바로 뭐라고 했냐면 "unit 9번에 있는 파일 test.in을 읽는데 읽어보니 파일을 끝까지 읽었다." 라고 한다. 프로그램에서 unit 9번에서 a라는 변수 값을 읽으라고 했는데 읽어볼려고 하니 파일은 없으니 당연히 에러 메시지를 뱉고 있다. 그러면 적당한 편집기로 파일에 내용을 넣어보자. 필자는 9.이라는 값을 가지고 있는 파일을 만들것이다. 아마 파일을 만들면서 보면 여러분이 실행 파일을 만든 디렉토리에는 이미 test.in이라는 파일이 존재하고 있을 것이다. 물론 내용은 없지만. 왜 그럴까 ? 그 이유는 좀전에 실행이 된 실행 파일 때문에 그렇다. 기본적으로 open 명령문에 의해 파일이 생성이 된 것이다. 그렇지만 현재 파일을 열어서 내용을 읽어 값을 대입해야 되는데 대입할 값이 없으므로 에러를 발생시킨 것이다. 그리고 test.out 파일을 열어보자.

```
F:\work>more test.out
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
        9.000000

F:\work>
```

아주 잘 저장이 되어 있다. 좀전에 open문의 문법에 보면 상당히 복잡한 옵션으로 구성이 되어 있는데 여기서 그런것을 설명한다는 것은 너무 힘든 일이고 기회가 닿으면 한번 DVF(Digital Visual Fortran) 도움말을 찾아서 보기 바란다. 한가지는 꼭 얘기를 해야되는데 OPEN는 데이터 선언문의 맨 마지막에 있어야 된다. 그러니 모든 데이터와 관련된 선언이 끝나고 난 후에서 OPEN을 사용한다고 알고 있어야된다.

자 그럼 read와 write에서 사용하는 unit이 어떻게 만들어지는가는 설명을 끝냈고 이제 format에 대해 알아보자. 이 format문은 현재 사용자가 출력하고자 하는 변수를 어떻게 출력할 것인가에 대한 정보를 담고 있다.

Default Format for List-Directed Output	
Data Type	**Output Format**
Integer	Iw
Character	Aw
Real, Double precision	Fw.d
Real, Double precision	Ew.d
Real, Double precision	Dw.d

여기서 Data Type은 출력하고자 하는 변수의 타입을 얘기하는 것이고 Output Format은 변수를 출력하는 지시자이다. 크게 나누면 I, A, F로 나눠진다. 각각 정수, 실수, 문자열 출력을 위한 지시자인데 각각의 지시자 다음에 붙어 있는 w는 출력하려는 변수의 크기를 의미한다. 즉, 8이라는 숫자를 출력하기 위해서 I를 그냥 쓰는 경우에는 출력되는 변수의 크기를 정하지 않고 기본적인 형태로 출력을 하라는 것이 된다. 그러면 기본적이 자리를 차지한 형태로 ‘ 8’이 출력이 되고 I1라고 하면 아무것도 출력이 되지 않는다. 이러한 현상은 윗장에서 이미 얘기를 했었다. 그렇지만 이러한 내용을 파일로 출력하는 경우에는 이상없이 출력이 된다. 지금부터는 파일에 출력이 되는 것을 가지고 얘기를 하겠다. I2로 출력을 하라고 하는 경우에는 ‘8’로 출력이 된다. 출력하려는 변수의 갯수가 정해진 것보다 작은 경우에는 ‘*’가 출력이 되면서 큰 변수를 현재 정해둔 자릿수로는 출력할 수 없다는 것을 보여준다. 또한 자릿수가 남는 경우에는 남는만큼 빈칸을 출력하고 값을 출력한다. 이러한 것은 I, A, F의 모든 경우에서 동일하다.

그런데 위에서 보면 d라는 숫자가 사용이 되었는데 이것은 모두 실수에 적용이 되는 사항이다. w만큼을 자릿수를 잡아두고 소숫점이하의 숫자를 얼마만큼 출력할지를 결정해 주는 것이다. 10.1234라는 숫자를 출력하는데 F10.5라고 하면 10개의 자리를 확보하고 소숫점 이하 5개의 자리를 확보한다. 즉, 정수부분을 출력하기 위해서 4개의 자리 소숫점 하나, 소숫점 5개의 값을 출력하게 된다. 좀전에 얘기했듯이 자리가 큰 경우 정수부분은 빈칸으로 처리를

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

하고 소숫점 이하 부분은 0의 값을 할당한다. 결과적으로 다음과 같이 '␣␣10.12340'이 출력이 된다. F나 D 지시자의 경우는 위에서 설명한 것을 동일하게 적용하고 약간 사용에서 차이점이 있지만 일단은 거의 같은 것이라고 생각하면 될꺼 같다. D는 Double precision의 약자로서 좀더 정확한 값을 얻고자 할때 사용한다. 그러나 E의 경우에는 exponential로 출력하기 때문에 약간 틀려진다. 위에 사용된 예제를 E10.5로 출력을 하는 경우 소숫점 이하 5개를 출력하고 E라는 기호를 사용하고 exponential의 +와 -를 표시하기 위해 한자리를 사용하며 그 크기를 나타내기 위해 2자리를 사용한다. 즉, E를 포함해서 4개의 자리를 사용하게 되고 '0'과 '.'을 출력하게 된다. 그렇게 되면 기본적으로 6개의 자릿수가 필요하게 되고 소숫점이하 5개를 출력하라고 했으니 10개의 자릿수로는 부족하게 된다. 이런경우에는 0 값을 출력하지 않고 '.10123E+02'라는 값이 출력되게 된다. 그러므로 표기할 수 있는 만큼 표기를 하고 4라는 값은 짤라버린다. 그러면 이제 E15.5로 출력을 하라고 하면 11개의 자리와 4개의 빈칸을 남겨둔 형태로 출력이 되는 것이다. '␣␣␣␣0.10123E+02'과 같이 출력이 된다.이런 것들 말고 데이터를 출력하는데 사용하는 지사자가 몇가지가 더 있지만 그런것은 여러분들이 직접 메뉴얼을 보면서 익히기 바란다.

이제는 데이터를 표현하는 것 외에 다른 포멧 지시자들을 살펴보자. 두 가지 정도만 더 알면 될꺼 같다. 'nX'와 '/'이다. X 지시자는 칸을 띄우면 지시자인데 앞에 붙어 있는 n은 그 갯수를 의미한다. 그리고 / 의 경우에는 줄을 띄우라는 의미이다. 이것의 경우에는 데이터를 읽을 때는 다음줄에 있는 데이터를 읽으라는 의미이고 출력을 하는 경우에는 다음 줄에 출력을 하라는 의미이다. 마지막으로 모든 포멧 지시자 앞에 숫자를 붙이는 경우에는 그 숫자만큼 반복을 하라는 의미이다. 이것으로 포멧문을 만드는 방법에 대해서는 어느정도 알게 되었을 것이다. 그러면 이제 간단한 예제를 한번 보자.

```
      program test

      character*20 a
      integer i, j
      real b, c

      open(unit=10,file='test.out')

      a='Hi ! I am Taechul Lee'
      i=1
      j=1000
      b=1.0
      c=1.e+3

      write(10,10) a, i, j, b, c
   10 format(a20, 1x, 2i10, /, 2f10.4)
      write(10,20) b, c
   20 format(2d10.5)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      write(10,30) b, c
   30 format(2e10.5)
      write(10,40) b, c
   40 format('Results are ', 2(f10.5,1x))

      stop
      end
```

이 프로그램을 실행하면 다음과 같은 결과를 얻을 수 있다.

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------
Hi ! I am Taechul Le          1      1000
    1.0000 1000.0000
.10000D+01.10000D+04
.10000E+01.10000E+04
Results are    1.00000 1000.00000
```

a라는 변수는 10개의 자리에 표시를 하라고 하였으므로 위에 나타나 있듯이 'Lee'라는 문자역을 다 표시 못했다. 그 다음 한칸을 띄우고 출력하고 하였으므로 i 변수는 31번째 칸에 표시가 되었고 또한 c변수도 41번째 칸까지 표시가 되었다. 그 다음 다음줄로 출력하는 것을 바꾸고 그 다음에 10칸을 확보한 다음 4개의 소숫점 이하 자릿수를 잡고 각각 '.0000'를 붙인 형태로 출력을 하였다. 그 다음 출력부분은 b와 c를 출력하는데 각각 d와 e 지시자를 이용해서 출력하였다. 그런데 잘보면 이 숫자들을 출력하다보면 두 출력부분이 붙어있어서 구분이 힘들다. 한칸씩의 구분을 넣었어야 하는데 그렇게 하지 못해서 위와 같은 결과를 얻는 것이다. 그렇지 않으면 출력할 칸을 더 확보해서 아예 칸이 벌어지게 했어야 하는데 그렇게 하지 못해서 생긴 일이다. 마지막 출력 부분은 굳이 문자열 변수를 할당하지 않고 직접 포멧문에 문자열을 넣은 예제이다. 이부분에서는 format문에 출력하고자 하는 문자열을 넣고 그 다음 출력을 한다. 그리고 '( )'를 이용해서 괄호안의 지시자를 반복하는 예를 보여주고 있다. 비슷한 출력을 계속하는 경우 지시자 앞에 숫자를 넣어서 반복하는 예 말고도 괄호를 이용해서 일정한 포멧을 반복할 수도 있다.

B.2.4 Redirection

입력과 출력에 대한 마지막 얘기를 시작해 보자. 이미 다음과 같은 프로그램을 이용해서 파일로부터 데이터를 읽어들이고 또한 결과값을 파일에 기록할 수 있다는 것을 배웠다.

```
      program test
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      open(unit=9,file='test.in')
      open(unit=10,file='test.out')

      read(9,*) a
      write(10,*) a

      stop
      end
```

위와 같이 만든 파일을 실행파일을 만들어서 실행하는 경우 만약 test.in이라는 파일에 1는 값이 있는 경우 다음과 같은 결과를 얻을 수 있다.

```
c:\>df test.for
DIGITAL Visual Fortran Optimizing Compiler Version 6.0
Copyright (C) 1997,1998 Digital Equipment Corp. All rights reserved.

test.for
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

/subsystem:console
/entry:mainCRTStartup
/ignore:505
/debugtype:cv
/debug:minimal
/pdb:none
C:\DOCUME~1\tclee\LOCALS~1\Temp\obj80.tmp
dfor.lib
libc.lib
dfconsol.lib
dfport.lib
kernel32.lib
/out:test.exe

c:\>test

c:\>type test.in
1
c:\>type test.out
   1.000000
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
c:\>
```

위의 결과는 프로그램의 컴파일부터 결과를 보여주는 모든 부분을 담고 있다. 자 그러면 이제 프로그램의 결과를 화면에 출력하는 예제를 만들어 보자. 이 예제는 이미 많이 했던 것이다.

```
      program test

      read(*,*) a
      write(*,*) a

      stop
      end
```

좀전에 작성한 프로그램과 거의 동일하지만 open문이 없어졌고 그 대신 입력과 출력을 기본값을 사용하도록 작성하였다. 이프로그램을 실행하면 다음과 같은 결과를 얻는다.

```
c:\>df test.for
DIGITAL Visual Fortran Optimizing Compiler Version 6.0
Copyright (C) 1997,1998 Digital Equipment Corp. All rights reserved.

test.for
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

/subsystem:console
/entry:mainCRTStartup
/ignore:505
/debugtype:cv
/debug:minimal
/pdb:none
C:\DOCUME~1\tclee\LOCALS~1\Temp\obj86.tmp
dfor.lib
libc.lib
dfconsol.lib
dfport.lib
kernel32.lib
/out:test.exe
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
c:\>test
2
   2.000000

c:\>
```

입력 변수의 값으로 2라는 값을 입력을했고 화면에 2.00000이라는 결과 값을 보여준다. 그렇다면 이 값을 저장할려면 어떻게 해야될까?

```
   2.000000
```

즉, 이 결과값을 적당한 파일로 저장을 할려면 어떻게 할 것이냐는 거다. open문을 이용해서 파일을 열어서 쓰겠다고 ? 그러면 다시 프로그램만들고 또 컴파일해야 되는데 ? 그런건 상관없다고 얼마걸리지 않으니까 ? 아니다. 이제부터 리다이렉션이 시작된다.

리다이렉션이라는 프로그램의 실행시 입력과 출력을 바꿔주는 것을 나타낸다. 입력이 키보드이고 출력이 화면인 경우 화면에 출력되는 결과를 적당한 파일로 얻기 위해서는 리다이렉션을 사용해야만 된다. 이 리다이렉션은 대부분의 경우 다 사용할 수 있다.

```
c:\>df test.for > compile.out
```

이렇게 입력을 하면 compile.out이라는 파일에 컴파일 결과가 들어 있다.

```
c:\>df test.for > compile.out
```

compile.out이라는 파일의 내용을 한번 보자.

```
c:\>type compile.out
DIGITAL Visual Fortran Optimizing Compiler Version 6.0
Copyright (C) 1997,1998 Digital Equipment Corp. All rights reserved.

test.for
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

/subsystem:console
```

```
/entry:mainCRTStartup
/ignore:505
/debugtype:cv
/debug:minimal
/pdb:none
C:\DOCUME~1\tclee\LOCALS~1\Temp\obj89.tmp
dfor.lib
libc.lib
dfconsol.lib
dfport.lib
kernel32.lib
/out:test.exe

c:\>
```

이제 뭔가 감이 잡히는가 ? 간단히 얘기를 하면 프로그램이 실행이 되고 그 결과 값을 저장하기 위해서는 ‘>’를 사용한다. 또한 비슷하게 입력값이 들어있는 파일을 지정해 주기 위해서는 ‘<’를 사용한다. 그러면 위의 프로그램은 입력과 출력을 기본값을 사용하였는데 이것을 sample.in과 sample.out으로 바꿔보자.

```
c:\>test < sample.in > sample.out
```

이 명령어를 입력하고 나면 아무런 얘기 없이 조용히 그냥 다시 사용자의 명령을 기다린다. 이렇게만 하면 좀전에 프로그램이 기본 입출력 값을 사용한거에 비해서 위의 명령은 sample.in과 sample.out이라는 파일을 이용해서 입력과 출력 파일로 이용하게 된다.

그런데 여기서 좀 주의할 것이 있는데 sample.in이라는 파일을 그냥 작성해서는 안된다. 윈도우 시스템에서 리다이렉션을 사용하는 경우에는 꼭 값을 입력하고 마지막에 엔터를 한번 더 넣어줘야 된다. 다음을 보자.

```
c:\>type sample.in
2

c:\>type sample.out
   2.000000

c:\>type test.in
1
c:\>
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

리다이렉션을 위해 사용했던 sample.in인 경우의 값을 보면 2라는 값과 엔터가 들어있어 한 줄이 더 들어간다. 또한 sample.out을 봐도 동일하다. 그렇지만 test.in이라고 open문을 이용해서 사용하는 경우에는 그러한 추가적인 엔터가 들어있지 않다. 이점을 주의하면 된다. 그리고 이러한 문제는 유닉스를 사용하는 경우에는 없다. read나 write는 한번 실행이 되고 난 후에는 커서를 다음 칸으로 넘긴다. 이 문제는 예전에 이미 설명을 했다. 하여간 더 이상 설명을 하지 않고 리다이렉션을 이용하는 경우에는 꼭 엔터를 하나 더 넣은 형태로 사용해야 된다는 것을 기억하자.

B.2.5 Dimension

그럼 이제 마지막 데이터의 형태로서 Dimension에 대해서 알아보자. 어짜피 공학 계산이라는 것을 하다가보면 행렬을 많이 사용할 수 밖에 없다. 일단 행렬에 대한 얘기를 하는 것보다 왜 필요한 가를 따져보자.

만약 여러분에게 3개의 데이터 값을 가지는 변수를 만들라고 하면 어떻게 하겠는가 ? 간단히 a, b, c 이런식으로 만들어서 저장을 할 것인가 ? 아니면 좀 머리를 써서 a1, a2, a3 이런식으로 만들 것인가 ? 그러면 만약에 100개의 데이터 값을 가지는 변수를 만들라고 하면 a1, a2,..., a100 이렇게 만들며 되지 않냐라고 생각할지 모르겠는데 이렇게 변수를 만들려면 얼마나 힘들겠나. 이럴때 사용하라고 Dimension이라는 것이 있다. 선언 방법에 대해 알아보자.

```
dimension a(100), b(10,10)
integer i(10)
real c(20)
```

위의 예제와 같이 크게 두가지 방법으로 dimension을 선언할 수 있다. 처음에 있는 것은 dimension이라는 키워드를 사용해서 a라는 변수는 100개의 요소를 가지는 행렬이라고 선언한 것이다. 그리고 b라는 변수는 다차원 행렬로 10×10 행렬이된다. 행렬의 처음 값은 1부터 시작을 한다. 그러니까 a변수는 a(1), a(2), ..., a(100)이 된다. 그리고 예전에 많이 쓰던 2차 행렬 아니면 3차 행렬이라고 하는 것들이 다차원 행렬이 된다. 이렇게 선언할 때에는 기본적인 변수의 타입에 따라 값이 결정된다. 처음에 얘기 했지만 아무런 얘기가 없는 경우에는(implicit문을 사용하지 않는 경우) i-n으로 시작하는 변수만 정수형이고 나머지는 실수형이라고 했다. 그러므로 a와 b는 모두 실수형 변수가 된다. 그리고 이렇게 dimension을 사용하지 않고 변수형의 선언하면서 행렬을 선언할 수도 있다. i라는 변수는 정수형 변수이미 10개의 구성요소를 자기는 것이 된다. 또한 마지막으로 c는 20개의 요소를 가지는 실수형 행렬이 되는 것이다. 이렇게 행렬을 선언할 때는 주의해야 될 것이 언어마다 시작하는 값이 다르다는 것이다. C로 프로그램을 작성하는 경우에는 0부터 n-1까지 요소가 존재하게 된다. 그러므로 각 언어마다 사용하는 것이 다르므로 각각의 언어에 맞춰서 사용해야만 된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

B.2.6 데이터 초기화

지금까지 여러가지 데이터 선언방법에 대해서 배웠다. 그러면 이러한 데이터 들이 어떻게 초기화가 되는지 알아보자. 그렇지만 중요한 것은 데이터를 선언하고 사용하기전에 이미 먼저 초기화를 꼭 시켜줘야만 한다. 그렇지 않고 사용하는 경우에는 문제가 발생할 소지가 있다는 것을 기억하기 바란다.

- 변수 : 0으로 초기화
- 배열 : 0으로 초기화

이러한 기본적인 초기화 방법외에 이미 배운 parameter문에 의한 경우에는 정해준 값으로 초기화가 된다. 이 parameter의 경우에는 상수를 선언하는 경우 유효하다. 그러므로 한번 선언을 하고 난 후에는 다시 다른 값을 대입할 수 없다.

```
program test

parameter  (n=2)
real realarray(10)
dimension realex(10)

do i=1,10
write(*,*) realarray(i), realex(i)
enddo

n=10

write(*,*) nroot, ttt

stop
end
```

위의 프로그램에 대해서 한번 살펴보자. 일단 n이라는 변수를 parameter문에 의해 2라는 값으로 선언을 하고 realarray 배열은 real 선언을 앞에 붙여서 선언을 하였고 realex 배열은 dimension 선언을 이용해서 선언을 하였다. realex도 또한 변수 이름에 따른 형선언 방법에 의해 real 변수로서 선언이 될 것이다. 그러므로 첫번째 DO문에 의해 출력이 되는 값은 real 형태의 변수가 모두 0으로 정해져서 나온다. 그 다음 parameter에 의해 선언된 n이라는 변수에 다시 값을 대입하였다. 이 경우 문제가 생긴다. 왜냐면 이 변수는 상수변수로서 선언이 된 것이기 때문에 다시 값을 대입할 수 없는데 대입하였으므로 에러를 발생할 것이다. 그리고 마지막에 특별한 선언 없이 두 변수를 사용하였다. 이렇게 사용하는 것은 상당히 않좋은 프로그램 방법인데 일단 예로서 보면 두 변수 모두 0으로 선언이 되어 출력될 것이다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

이번에는 이렇게 프로그램이 제공하는 기본적인 0으로 정하는 것 말고 다르게 값을 대입하는 것을 알아보자. 이런 경우에는 DATA 라는 선언문을 사용해서 값을 정해줄 수 있다.

- 변수의 갯수와 초기화 값의 갯수가 일치해야만 한다.
- 모든 실행문 앞에 나와야만 된다.
- 배열의 이름을 그냥 사용하는 경우는 모든 요소를 나타낸다.
- 배열의 값을 초기화 하는 경우에는 implied do 문을 사용할 수도 있다.
- 동일한 초기값을 사용하는 경우에는 그 갯수를 정해서 곱의 형태로 나타낼 수 있다.

기본적인 사용방법은 다음과 같다.

```
DATA var-list /data-list/
```

그러면 사용하는 방법에 대해 예제로서 알아보자.

```
      program test

      parameter  (n=2)
      real a, b, realarray(10)
      dimension realex(2,10)
      data a, b /10.0, 2./
      data realarray /1.,2.,8*10./
      data (realex(1,i),i=1,10) /10*1./
      data (realex(2,i),i=1,4) /2*(12.1, 13.1)/
      data (realex(2,i),i=5,10) /5*2./

      stop
      end
```

위의 예제에서 변수 a와 b는 각각 10.0과 2.0으로 초기화가 되었다. 그 다음 realarray라는 배열은 1.0과 2.0, 나머지는 모두 10.0으로 초기화가 되어있는 상태이다. 그 다음 realex배열의 처음 1행은 모두 1.0으로 초기화가 되고 두번째 행에서 처음 네개는 각각 12.1과 13.1로 두번 초기화를 했다. 마지막 5개는 2.0으로 초기화를 한 상태이다. 이렇게 다양한 방법으로 초기화를 할 수 있다는 것을 알고 있어야 된다.

마지막으로 알아야될 데이터과 관련이 된것은 COMMON문인데 이 부분은 뒤에서 부프로그램 부분에서 따로 다룰 것이므로 자세한 얘기는 하지 않겠다.

지금까지 설명할 것을 정리해 보자.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- Implicit 문을 이용하면 기본적인 변수이름에 따른 데이터 형을 결정해 줄 수 있다.
- Parameter 문을 사용해서 상수의 값을 정해줄 수 있다.
- Data 문을 이용해서 변수의 값을 초기화 시켜줄 수 있다.
- Common을 사용하는 경우 메인 프로그램과 부프로그램이 같은 변수값을 사용할 수 있다.

B.2.7 Summary

이번 장에서 배운 것을 정리하면 다음과 같다.

- 데이터 선언방법(integer, real, character, double precision)
- 입출력문 작성요령
- open 사용방법
- format문 작성요령
- 각각의 지시자 사용 방법(I, F, E, D, A, X, /)
- 행렬을 만드는 방법
- 각각의 데이터 초기화 방법 및 형선언문과 관련된 여러가지 지시어들

B.3 제어문

이미 계속해서 얘기를 해 왔지만 언어를 배우는데 있어서 가장 중요한 것 가운데 하나가 제어문이다. 그리고 지금까지 배운 지식을 바탕으로 볼때 포트란이라는 언어는 상당히 엄격한 언어이다. 컬럼을 맞추지 않는다든지 변수명을 잘못 사용한다던지 하면 어김없이 에러 메시지나 경고 메시지를 뿌린다. 제어문에 대한 것도 마찬가지지만 그렇게 엄격한 문법을 사용하기 때문에 한번 배우면 쉽게 사용할 수도 있다.

B.3.1 STOP문

STOP문은 프로그램을 종료시키는 명령이다. 보통 END 전에 사용된다. 주 프로그램에서는 STOP을 사용하면 부프로그램에서는 RETURN을 사용한다. 지금까지 만들어온 프로그램들은 모두 주 프로그램으로 program이라는 키워드로 시작을 한다. 부프로그램 얘기를 잠시했

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

는데 이 부분은 뒤에서 좀더 심도 있게 다룰것이다. 그리고 그 프로그램 실행의 종료를 의미하는 STOP과 그리고 프로그램 작성이 끝난것을 의미하는 END로 구성이 되어 있다. 간혹 이 STOP을 생략했다가 고생을 하는 경우가 있는데 (필자의 경우다. 석사에 프로그램을 작성하다가 빼먹고는 디버깅에 밤을 세웠다.) 꼭 넣어줘야만 프로그램이 종료된다. 프로그램 실행이 끝이나면 다음과 같은 문장을 화면에 보여준다.

```
Stop - Program terminated.

c:\>
```

이러한 말은 STOP문에 의해 발생하는 것이다.

B.3.2 GOTO문

이 GOTO는 악명이 대단한 명령문이다. 이것을 사용한다는 것때문에 지금까지 포트란이 Structured Language라는 얘기를 못들어 온 것이다. 이 명령문을 한글로 표현하면 무조건적 분기문이라고 하는데 이말이 의미하는대로 마음대로 언제나 어디로나 갈 수 있는 명령문이다. 이렇게 얘기를 하면 상당히 좋은 것으로 생각이 들지 모르겠지만 한번 생각을 해 봐라. 누군가 프로그램을 짜 두었는데 언제 어디서 어떻게 갈지 모르고 그냥 프로그램의 이곳 저곳으로 가는 프로그램을 여러분이 이해할려면 얼마나 힘들겠나. 이 GOTO문은 대부분 IF문으로 대체할 수 있으므로 IF문을 써서 작성하기 바란다. 사실 이러한 명령문이 있다는 사실 조차 여러분에게 말하고 싶지 않지만 그래도 잠시 설명을 해야 될꺼 같아서 넣은 것이다. 절대 사용하지마라. 좀 길게 프로그램을 작성하더래도 절대 절대 사용하지 말아야될 명령문이다.

B.3.3 PAUSE문

PAUSE 명령은 프로그램을 일시 정지 시키기 위해 사용이된다. 프로그램을 작성하는 경우 갑작스래 이상한 값이 나오는 경우 프로그램을 계속 수행할지 아니면 중지 시킬지를 사용자가 선택할 수 있도록 도와 준다.

```
Fortran Pause - Enter command<CR> or <CR> to continue.
```

이러한 메시지를 보여주고 사용자의 입력을 기다리고 있게 된다.

B.3.4 CONTINUE문

CONTINUE 문은 일반적으로 DO 제어문의 연산을 계속하기 위해 사용되었지만 대부분 계산 과정을 계속하기 위해서 사용이된다. 주로 문번호를 붙여서 사용하게 되며, 좀 뒤에 산술

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

IF문을 설명하는 예제에서 사용하게 된다. 그러나 요즘에는 프로그램 작성시 거의 사용되지 않는다.

B.3.5 CALL문

포트란에서는 부프로그램이라고 SUBROUNTINE과 FUNCTION이 있는데 FUNCTION은 값을 돌려 받기 위해서 사용이 되고 SUBROUTINE은 일련의 계산을 수행한 후 값을 돌려주지는 않는다. 이러한 SUBROUTINE을 수행하기 위해서는 CALL 명령을 붙여서 사용한다. 정확하게 얘기를 한다면 계산 결과를 넘겨주긴 넘겨주데 명확한 방법을 사용해서 넘겨주는 것이 아니다. 나중에 이 부분에 대해서 직접 다룰 것이다.

```
CALL SUBROUTINE
```

형식으로 사용한다.

B.3.6 RETURN문

위에서 메인 프로그램을 종료하기 위해서는 STOP문을 사용한다고 했었다. 그리고 메인 프로그램에서 부프로그램 가운데 하나인 SUBROUTINE을 부르기 위해서는 CALL을 사용한다고 했다. 그러면 이러한 SUBROUTINE이 종료되고 다시 메인 프로그램으로 돌아가기 위해서는 무엇을 사용해야 할까 ? 바로 이 RETURN을 사용하면 된다. 즉, 프로그램의 수행 제어를 메인으로 돌리기 위해서 사용되는 명령이다.

```
SUBROUTINE TEST(a,b)
    .
    .
    .
RETURN
END
```

일련의 과정을 마친후 RETURN으로 제어를 넘기고 프로그램이 끝났다는 의미의 END를 사용하면 된다. 자세한 부프로그램의 작성은 나중에 따로 설명을 하겠다.

B.3.7 IF문

IF문은 크게 산술, 논리, 블록 등으로 구분이 되지만 여러분들이 많이 사용할 부분은 논리와 블록일 것이다. 산술 IF 문은 계산된 결과에 따라서 각각의 할당된 문 번호를 실행하라는 것

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

인데 필자의 경험으로 거의 사용하지 않는다. 왜냐면 굳이 이런 산술 IF를 사용하지 않아도 논리 IF로서 구현할 수 있고 또한 이 산술 IF를 사용하면 프로그램을 읽기가 쉽지 않기 때문이다. 그렇지만 일단 있다는 사실과 예제에 대해서 배워보자.

```
      program test

      real a, b

      a=10.0
      read(*,*) b

      if(a-b) 10, 20, 30

   10 write(*,*) 'A is less than B'
      goto 40
   20 write(*,*) 'A is equal to B'
      goto 40
   30 write(*,*) 'A is greater than B'
   40 continue

      stop
      end
```

왜 이 산술 IF문을 사용하면 좋지 않을지는 조금전에 설명한 GOTO문을 사용했기 때문이다. 그리고 예전에 이미 얘기를 했지만 실수를 비교하는 것에서 같다는 표현이 별로 않좋다는 것에 대해서 논의를 했었다. 자 한번 보자. 일단 산술 IF는 IF문 다음에 나오는 결정에 따라 각각의 문번호에 할당되는 곳으로 이동을 한다. 'a-b'의 결과가 어떻게 되냐에 따라 각각의 문번호에 대한 하는 곳으로 이동을 해서 그 문번호에 대한 것을 실행한다. 뒤에 3개의 문번호가 나오는데 계산 결과 값이 각각 작거나 같거나 크거나에 따라 옮겨가며 실행이 된다. 그런데 그 다음 goto문이 없으면 계속해서 실행을 하게 된다. 무슨 얘기냐면 위의 예제에서 'goto 40'이 없다면 10번 문장이 실행이 된 후 20번 문장과 30번 문장이 실행이 되어 버린다. 그러므로 10번 문장이 실행 된 후에서 다시 자리를 옮겨서 40번으로 가야만된다. 또한 실수를 비교하는 경우에 사용하지 말라고 했는데 물론 비교를 아예 하지 말라는 얘기는 아니다. 단, 같다는 비교를 특히 정수와 하지 말라는 것이다. 특히 여러번 복잡한 계산과정이 있는 프로그램의 경우에는 error가 쌓여서 1.0으로 표시 되던 숫자의 값이 0.9999999의 값이 들어가는 경우도 있기때문에 크거나 작거나를 비교하는 경우는 괜찮지만 같다는 비교를 해서는 엉뚱한 결과를 얻는 경우도 있다. 위의 예제는 문제가 전혀 없으나 만일 일련의 계산과정을 거친 후에는 엉뚱한 값이 들어 있을 수 있기 때문이다.

이제 원래 사용하는 논리 IF문에 대해서 배워보자. 보통의 경우에는 IF 다음에 판단의 기준이 나오고 기준에 따라 수행할 명령을 기술한다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
IF (logical_expression) statement
```

여기서 logical_expression은 다음과 같은 것들이 있다.

Relational & Logical Expression	
Operator	**Meaning**
.LT. or <	Less Than
.LE. or <=	Less than or Equal to
.EQ. or ==	Equal to
.NE. or / =	Not Equal to
.GT. or >	Greater Than
.GE. or >=	Greater than or equal to
.AND.	True if both A and B are true
.OR.	True if either A, B, or both, are true
.NEQV.	True if either A or B is true, but false if both are true
.XOR.	Same as .NEQV.
.EQV.	True if both A and B are true, or both are false
.NOT.	True if A is false and false if A is true

이러한 판단 기준을 만족 시켜 logical_expression이 참인 경우에만 statement를 실행을 한다. 예를 들어보자.

```
program test

read(*,*) i

if(i .eq. 1) write(*,*) "Right !! You enter 1"

stop
end
```

아주 간단히 만든 예제로 사용자가 1인 값을 i에 입력을 하는 경우에만 write문이 실행이 되고 아닌 경우에는 아무것도 하지 않고 끝을 낸다. 이렇게 IF을 사용하는 경우는 간단히 무언가를 테스트 한 후 간단한 출력이나 계산의 변경을 원하는 경우에나 사용한다. 더 복잡한 명령을 실행하기 위해서는 THEN이나 ELSE를 사용하는 IF문을 작성해야만 한다.

그러나 여러개의 명령을 수행하는 경우에는 THEN이라는 키워드를 사용해야만 되는데 이러한 경우에는 블록 IF라고 부른다. 기본 형태를 보자

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
IF (logical_expression) THEN
   statement block
ELSE IF (logical_expression) THEN
   statement block
ELSE
   statement block
ENDIF
```

저번 장에서 배운 matlab의 경우와 거의 유사하다. 이렇게 블록 IF를 사용하는 경우에는 꼭 마지막에 ENDIF를 붙여줘야만 된다. 예를 들어보자. 처음에는 간단한 예제를 들어보고 다음에는 좀전에 배운 산술 IF문을 바꿔서 해보자.

```
      program test

      real a

      read(*,*) a

      if(a .gt. 0.) then
         write(*,10) a
      else
         write(*,20) a
      endif

   10    format('A is positive and the value is ', f10.3)
   20    format('A is negative or zero and the value is ', f10.3)

      stop
      end
```

위의 예제는 간단히 a라는 값에 따라서 양수인지 아니면 음수인지를 판단 값을 출력하는 프로그램이다. 포멧문은 어디에 있던지간에 프로그램이 종료하기 전에 있기만 하면 된다.

```
      program test

      real a, b, c

      a=10.0

      read(*,*) b
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
        c=a-b

        if(c .lt. 0.) then
           write(*,*) 'A is less than B'
        else if(c .eq. 0)
           write(*,*) 'A is equal to B'
        else
           write(*,*) 'A is greater than B'
        endif

        stop
        end
```

이 예제는 산술 IF문을 블록 IF로 바꾼 것이다. 이렇게 프로그램을 작성하는 것이 더 이해하기 쉽고 GOTO문을 사용하지 않아서 좀더 구조적 프로그래밍을 할 수 있다.

B.3.8 DO문

DO문은 계산을 반복적으로 실행하는 경우에 사용하는데 matlab에서 얘기한 for나 while과 유사한 명령어이다.

```
  DO [label[, ] ] [loop-control]
     block
  [label] term-stmt
```

위와 같이 기본형이 주어지는데 DO 다음에 라벨을 표시하고 그리고 얼마나 반복수행할지를 결정해 준다. 자 예를 보자.

```
        program test

        real sum
        integer i

        sum=0.

        do 10 i=1,10
           sum = sum + i
     10 continue
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
write(*,*) sum

sum = 0.

do i=1,10
   sum = sum + i
enddo

write(*,*) sum

sum = 0.

do i=2,10,2
   sum = sum + i
enddo

write(*,*) sum

stop
end
```

처음에 사용한 DO문은 기본적인 문법에 충실하게 만든 것이다. do 다음에 문번호를 입력했으며 그 다음 몇번 돌지를 결정하는 i라는 변수를 사용했다. 여기서 변수 i는 1부터 시작을 해서 10까지 차례로 증가를 한다. 원래는 세개의 인수를 사용해서 시작값, 종료값, 증가값으로 사용하지만 증가값이 없는 경우에는 기본적으로 1을 사용한다. 그러므로 이 경우에는 1부터 10까지의 합을 구하게 된다. 그런데 이렇게 작성하는 경우에는 문번호를 하나 사용하고 continue명령을 사용해야만 하는 문제가 있다. 두번째 DO문은 enddo로 끝이나며 문번호를 사용하지 않았다. 여러분들은 어떻게 생각하는가 ? 이렇게 작성하는 것이 더 간결해 보이지 않는가 ? 하여간 문번호를 사용하다가 보면 좀 큰 프로그램을 작성하는 경우에는 혼란이 생긴다. 포멧문에 문번호를 지정하는 것도 번거럽고 어디에 어떤 문번호를 사용했는지도 체크를 해야 되는데 DO문 마저 문번호를 사용하면 더욱더 복잡하게 된다. 세번째 예제는 세번째 인수 즉, 증가값을 결정해주는 예제이다. 이번에 증가값은 2로 이 DO문은 2부터 시작을 해서 10까지 짝수로 증가를 하면서 그 값들의 합을 계산하는 예제이다. 이정도 하면 기본적인 DO문을 이해하는데 어려움은 없으리라 생각이 된다.

그러면 다중 DO문을 사용해 보자. 다중 DO문을 사용하는 경우는 대부분 행렬을 다루는 문제에서 사용하게 된다. 전장에서 이미 행렬을 만든 방법에 대해서는 설명을 하였다. 예를 들어 2×2 행렬의 덧셈을 해보자.

```
program test
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
real a(2,2), b(2,2), c(2,2)

do i=1,2
do j=1,2
   read(*,*) a(i,j)
enddo
enddo

do i=1,2
do j=1,2
   read(*,*) b(i,j)
enddo
enddo

do i=1,2
do j=1,2
   c(i,j)=a(i,j)+b(i,j)
enddo
enddo

do i=1,2
do j=1,2
   write(*,*) c(i,j)
enddo
enddo

stop
end
```

좀 지루해 보이지만 a와 b에 대해 각각 4개의 입력 값을 넣어주면 c 변수의 값을 얻을 수 있다. 변수를 입력하기 싫은 경우에는 open문을 이용해서 읽을 파일을 정해주고 난 후 값을 입력 받으면 된다. 그렇지만 그렇게 한 경우에도 데이터 파일의 형태가 알아보기 힘들 것이다. 이렇게 바꾼 경우 어떻게 되나 생각을 해보자.

```
program test

real a(2,2), b(2,2), c(2,2)

open(9,file='test.in')
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
do i=1,2
do j=1,2
   read(9,*) a(i,j)
enddo
enddo

do i=1,2
do j=1,2
   read(9,*) b(i,j)
enddo
enddo

do i=1,2
do j=1,2
   c(i,j)=a(i,j)+b(i,j)
enddo
enddo

do i=1,2
do j=1,2
   write(*,*) c(i,j)
enddo
enddo

stop
end
```

test.in 이라는 파일에서 계산에 사용될 데이터 값을 읽어들인다. 그러면 test.in 파일은 다음과 같이 준비해야된다.

```
1
2
3
4
5
6
7
8
```

이게 뭘까? 여러분들이 이 값을 보면 아무리 테스트용으로 만든 프로그램이라고 할지래도 좀 보기가 않좋지 않은가 ? 이렇게 입력하는게 싫으면 다음과 같이 하고 싶을지 모르겠다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
1 2 3 4 5 6 7 8
```

그런데 이렇게 만들고 나면 에러를 발생할 것이다. 여러분들은 분명히 8개의 데이터 값을 주었지만 프로그램은 데이터가 없다고 하면서 에러를 발생한다. 왜 그럴까 ? read나 write문은 한번 실행이 될때 특별한 지시자(줄바꿈 '/')가 없는 한 한줄의 데이터를 읽어 들인다. 그리고 그 데이터 들은 탭이나 빈칸으로 구분을 한다. 그런데 읽어만 들여도 대입할 변수가 없으면 버려버리기 때문에 첫줄에 있는 변수 가운데 1의 값이 a(1,1)에 할당이 되고 없기 때문에 데이터가 모자르다고 에러 메시지를 보여주는 것이다. 실제 가장 이상적으로 생긴 데이터 입력 형태는 아마도 다음과 같을 것이다.

```
1 2
3 4
5 6
7 8
```

실제 행렬의 모습과 유사한 이러한 모습으로 데이터를 입력하는 것이 보기에 가장 좋고 이해하기도 쉬울 것이다. 이렇게 입력을 할려면 어떻게 해야될까 ? 이런경우에 implied DO loop라는 것을 이용하면 된다. 즉, 내제된 DO 루프이다.

```
program test

real a(2,2), b(2,2), c(2,2)

open(9,file='test.in')

do i=1,2
   read(9,*) (a(i,j),j=1,2)
enddo

do i=1,2
   read(9,*) (b(i,j),j=1,2)
enddo

do i=1,2
   do j=1,2
      c(i,j)=a(i,j)+b(i,j)
   enddo
enddo

do i=1,2
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
   write(*,*) (c(i,j),j=1,2)
enddo

stop
end
```

이렇게 작성하면 DO 루프 하나가 없어지고 입력이나 출력 부분에 아예 DO 루프와 비슷한 형태의 것이 있게 된다. 그리고 read나 write가 한번씩 사용이 되었으므로 줄을 바꾸지 않고 계속 진행해 나간다. 이걸 이용한다면 한줄에 입력할 수도 있을 것이다.

```
program test

real a(2,2), b(2,2), c(2,2)

open(9,file='test.in')

read(9,*) ((a(i,j),j=1,2),i=1,2)
read(9,*) ((b(i,j),j=1,2),i=1,2)

do i=1,2
do j=1,2
   c(i,j)=a(i,j)+b(i,j)
enddo
enddo

write(*,*) ((c(i,j),j=1,2),i=1,2)

stop
end
```

그런데 이렇게 작성을 하면 원래 의도와는 다르게 좀 구분하기 힘들 것이다. 그러니 이것은 그냥 예제로서 보여준 것이므로 좀전의 것을 사용하는 것이 좋을 거 같다.

B.3.9 Summary

이번장에서는 각종 제어문에 대해서 배웠다. 이것들을 정리하면 다음과 같다.

- STOP문
- GOTO문

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- PAUSE문 사용방법
- CONTINUE문
- CALL문 사용방법
- RETURN문
- 산술 IF와 블록 IF문 작성 방법
- 각종 관계 및 논리 연산자 사용법
- 일반적 DO문과 다중 DO문, implied DO문 작성법
- 다차원 배열 작성법

B.4 부프로그램

지금까지 배운 것은 대부분은 프로그램 작성의 기본적인 요소가 되는 것에 대해 배웠다. 이번에는 이러한 요소를 이용해서 포트란에서 사용되는 부프로그램 작성 방법에 대해서 배운다. 부프로그램이라고 얘기하는 것은 FUNCTION과 SUBROUTINE이 있기 때문이다. 각각은 사용방법이 약간 다르다. 그리고 Matlab에서 이미 얘기를 했지만 프로그램은 작은 단위로 나눠서 메인 프로그램에서는 복잡해 보이는 연산과정 같은 것은 없애는 것이 더 효율적인 프로그래밍 기법이다.

B.4.1 FUNCTION

function은 가장 기본적인 부프로그램이다. function이 돌려주는 값은 대부분 변수하나이므로 여러가지 변수값을 받는 경우에는 부적절한다. 기본적인 형태는 다음과 같다.

```
Type FUNCTION function_name(arg1,arg2...)
statement1
statement2
statement3
.
.
function_name=returned values
return
end
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

일단 사용하고자 하는 함수가 돌려주는 값의 데이터 형태에 대한 Type이 필요하고 FUNCTION이라는 키워드 다음 함수의 이름을 적는다. 함수는 하나의 인수를 가질수도 있고 여러개의 인수를 가질수도 있다. 그 다음부터는 일반적 프로그램을 작성하면 되고 함수이름에 돌려줄 값을 명시한다음 꼭 RETURN문이 있어야 된다. 그 다음 ENDF로서 프로그램이 끝났음을 알려주면 된다. 간단한 예제를 보면서 작성 방법에 대해서 알아보자.

```
program test

real a, b, c

a=1.
b=2.

c=sum_test(a,b)

write(*,*) c

stop
end

real function sum_test(x,y)

sum_test=x+y

return
end
```

이 예제는 간단히 합을 계산하는 함수를 작성하였다. 이 함수가 돌려주는 값는 real 형태의 값이며, sum_test에서 사용된 인수인 x와 y는 지역변수로서 어떠한 이름으로 선언이 되어 있어도 무방하다. 그리고 return문이 나오기 전에 함수이름에 해당하는 sum_test에 주어진 두 변수의 합의 값을 대입하고 마지막으로 end로 프로그램은 끝이 났다. 현재는 너무 간단하기 때문에 이러한 함수의 작성 필요을 느끼지 못할지 모르겠지만 실제는 이렇게 간단한 프로그램을 작성하지 않는다. 함수를 이용할때는 단순히 이름과 이 함수가 돌려줄 변수만 선언이 되어 있으면 된다.

다른 예로서 더 많은 인수가 있는 경우는 어떻게 할지 생각을 해보자.

```
program test

real a, b, c, d, e, f, g
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
a=1.
b=2.
c=1.
d=2.
e=1.
f=2.

g=sum_test(a,b,c,d,e,f)

write(*,*) g

stop
end

real function sum_test(o,p,q,r,s,t)

sum_test=o+p+q+r+s+t
return
end
```

이 프로그램은 인위적으로 인수를 많이 쓰는 경우를 보여주고 있다. 이렇게 작성을 해도 되지만 이러한 경우에는 COMMON이라는 것을 사용하는 것이 더 좋다. 아직 이부분에서 대해서 자세한 설명을 하지 않았지만 간단히 알아보자. 두가지 측면에서 장점이 있는데 하나는 메모리 절약적인 면에서 아주 심한 낭비이다. 메인문에서 사용한 변수는 real 형태로 7개의 변수가 사용되었다. 또한 함수에서는 명확한 변수의 선언부분은 없지만 인수로 받는 모든 변수를 메모리에 할당해야지만 사용할 수 있다. 즉, 위의 sum_test 함수는 인수로서 받아들이는 변수 7개의 변수를 메모리에 할당해야만 작동이 가능하다. 그러므로 메모리를 낭비하게 된다. 또 하나는 이 COMMON을 사용하면 변수를 값을 어디서든지 바꿀 수 있다. 즉, matlab에서 사용되었던 global 변수와 비슷하다. 마지막으로 사용하는데 있어서 인수가 많아지므로 좀 복잡하다는 단점도 있다. 이것을 COMMON을 사용해서 간단히 표현하면 다음과 같다.

```
program test

real a, b, c, d, e, f, g
common /data/ a,b,c,d,e,f,g
a=1.
b=2.
c=1.
d=2.
e=1.
f=2.
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
g=sum_test()

write(*,*) g

stop
end

real function sum_test()
common /data/ a,b,c,d,e,f,g
sum_test=a+b+c+d+e+f+g
return
end
```

이렇게 하는 경우에는 아예 인수가 없는 형태의 함수도 작성이 가능하지만 사실 이런 식으로 사용하기 위해서 COMMON문이 있는 것은 아니다. 이 예제는 단순히 COMMON문을 사용하는 것을 보여주기 위해서 작성한 것이다.

자 이번에는 함수의 인수로서 배열이 들어오는 경우를 생각해 보자. 도대체 배열을 인수로 보낼때는 어떻게 해야될까 ? 예제를 보자.

```
program test

real a(10), sum_result

read(*,*) (a(i),i=1,10)

sum_result=sum_test(a)

write(*,*) sum_result

stop
end
```

지금 일단 함수부분에 대한 것은 빼고 인수로서 배열을 사용하는 경우를 생각해보자. 위와 같이 단순히 a라는 변수를 실수 배열로서 선언을 한 후 함수의 인수로서 넣어주기만 하면 사용할 수 있다. 그러면 함수에서 이 값을 받아서 적절한 계산을 할 수 있다. 즉, 배열이래도 일반 다른 변수와 차이 없이 메인 프로그램에서 사용하면 된다. 그럼 함수도 그럴까 ? 아니다. 함수에서는 약간 다른다. 함수는 현재 배열변수가 들어왔다는 것을 모르고 단지 어떠한 변수 하나만을 받은 상황이므로 배열이 인수로 들어온다는 것을 알려줘야된다.

```
real function sum_test(x)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
real x(10)

sum_test=0
do i=1,10
sum_test=x(i)
enddo

return
end
```

위에서 보는 것과 같이 인수로서는 x라는 인수가 들어왔다고 알려주지만 함수내에서 이 변수는 배열변수라는 것을 알려주기 위해서 다시 배열 선언을 해 줘야만 사용할 수 있다. 그 다음은 그냥 배열을 쓰듯이 계산에 사용만 하면된다.

B.4.2 SUBROUTINE

이번에는 부프로그램 가운데 하나인 서브루틴에 대해 설명을 한다. 서브루틴은 위에서 설명한 FUNCTION과는 약간 다른점이 있다. 첫번째로 SUBROUTINE은 FUNCTION과는 달리 결과값을 명확히 돌려주지는 않는다. 함수를 이용하는 경우에는 함수가 돌려주는 값을 적당한 변수에 대입해주기 때문에 사용자가 프로그램 작성시 어떠한 변수에 결과값이 대입되는지 확실히 알 수 있지만 서브루틴의 경우에는 부프로그램 실행시 CALL이라는 예약어를 이용해서 서브루틴을 부를뿐 어떤 변수가 입력 변수이고 어떻게 결과값이 돌아오는지 잘 모른다. 그렇지만 함수의 경우에는 돌려받을 수 있는 값이 하나 밖에 없지만 서브루틴은 그러한 제약이 없다. 물론 COMMON문을 이용하는 경우 이러한 제약이 없어지지만 그렇게 값을 바꾸는 것은 좋은 프로그램 방법이 아니다. 일단 기본 형태에 대해서 알아보자.

```
SUBROUTINE subroutine_name(arg1, arg2...)
statements
.
.
return
end
```

SUBROUTINE이라는 지시자를 붙인 후 서브루틴의 이름을 정해주고 각각의 인수를 정의한다. 그 다음 실행하고자하는 문장을 기술한 다음에 마지막으로는 RETURN문과 END문을 써주면 된다. 이러한 서브루틴을 호출하는 경우에는 이미 얘기했던 CALL을 이용해서 서브루틴을 부르면 된다. 그러면 간단한 예를 들어보자.

```
program test
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
real a, b, sum_result

a=1.0
b=2.0

call sum_test(a, b, sum_result)

write(*,*) sum_result

stop
end
```

이 예제는 세개의 인수를 받아서 일련의 계산을 하는 예제이다. 서브루틴을 호출하는 것은 call이라는 것을 사용했고 서브루틴의 이름은 sum_test이다. 그러면 이제 서브루틴을 만들어 보자.

```
subroutine sum_test(x,y,z)

real x, y, z

z=x+y

return
end
```

서브루틴은 간단하게 두개의 인수 합을 계산한다음 적당한 변수에 이 값을 치환해 주는 것이다. 서브루틴에서 사용된 x, y, z는 모두 지역변수이다. 위의 함수의 예제에서는 함수에서 사용하는 인수에 대한 데이터 형 선언부분이 없다. 필요없어서가 아니라 그냥 생략을 한 것이다. 배열의 예제에서는 분명히 인수가 무엇인지 선언을 해 주었다. 사실 위의 예제에서는 서브루틴에서 사용하는 인수에 대한 선언 부분이 필요 없지만 더욱 확실한 프로그램을 작성하는 경우에는 써주는 편이 더 좋다. 위와 같은 프로그램을 작성하는 경우에는 굳이 서브루틴을 이용할 필요는 없다. 이러한 경우에 사용하라고 서브루틴이 있는 것이 아니기 때문이다. 그러면 좀더 현실성 있는 예제를 만들어보자. 배열의 합을 계산하는 프로그램을 만들어보자. 일단 시작은 서브루틴을 사용하는 않는 경우로 해 보자.

```
program test

real a(2,2), b(2,2), c(2,2)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
do i=1,2
   read(*,*) (a(i,j),j=1,2)
enddo

do i=1,2
   read(*,*) (b(i,j),j=1,2)
enddo

do i=1,2
   do j=1,2
      c(i,j)= a(i,j)+b(i,j)
   enddo
enddo

do i=1,2
   write(*,*) (c(i,j),j=1,2)
enddo

stop
end
subroutine sum_test(x,y,z)

real x, y, z

z=x+y

return
end

program test

real a(2,2), b(2,2), c(2,2), d(2,2), e(2,2)

do i=1,2
   read(*,*) (a(i,j),j=1,2)
enddo

do i=1,2
   read(*,*) (b(i,j),j=1,2)
enddo
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
do i=1,2
   read(*,*) (c(i,j),j=1,2)
enddo

do i=1,2
   do j=1,2
      d(i,j)= a(i,j)+b(i,j)
   enddo
enddo

do i=1,2
   do j=1,2
      e(i,j)= c(i,j)+a(i,j)
   enddo
enddo

do i=1,2
   write(*,*) (d(i,j),j=1,2)
enddo

do i=1,2
   write(*,*) (e(i,j),j=1,2)
enddo

stop
end
```

이 프로그램을 이해하는 것은 그리 어려운 것이 아닐꺼다. 그런데 잘보면 계속 반복되는 부분이 있다. 일단 행렬의 합을 계산하는 부분을 서브루틴으로 작성해 보자.

```
subroutine sum_array(x,y,z)

real x(2,2), y(2,2), z(2,2)

do i=1,2
   do j=1,2
      z(i,j)= x(i,j)+y(i,j)
   enddo
enddo

return
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
end
```

이 서브루틴을 이용하면 위의 프로그램을 다음과 같이 수정해야된다.

```
program test

real a(2,2), b(2,2), c(2,2), d(2,2), e(2,2)

do i=1,2
   read(*,*) (a(i,j),j=1,2)
enddo

do i=1,2
   read(*,*) (b(i,j),j=1,2)
enddo

do i=1,2
   read(*,*) (c(i,j),j=1,2)
enddo

call sum_array(a,b,d)
call sum_array(c,a,e)

do i=1,2
   write(*,*) (d(i,j),j=1,2)
enddo

do i=1,2
   write(*,*) (e(i,j),j=1,2)
enddo

stop
end
```

자 이제 보면 sum_array라는 서브루틴이 무엇을 하는지 모르겠지만 하여간 프로그램이 많이 간결해졌다. 그러면 좀더 생각을 해보자. 행렬의 값을 부르는 부분도 좀 줄여보자.

```
subroutine read_array(x)

real x(2,2)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
do i=1,2
   read(*,*) (x(i,j),j=1,2)
enddo

return
end
```

그러면 메인을 다음과 같이 바꿔야 된다.

```
program test

real a(2,2), b(2,2), c(2,2), d(2,2), e(2,2)

call read_array(a)
call read_array(b)
call read_array(c)

call sum_array(a,b,d)
call sum_array(c,a,e)

do i=1,2
   write(*,*) (d(i,j),j=1,2)
enddo

do i=1,2
   write(*,*) (e(i,j),j=1,2)
enddo

stop
end
```

하는길에 출력하는 부분마저 수정을 해보자.

```
subroutine write_array(x)

real x(2,2)

do i=1,2
   write(*,*) (x(i,j),j=1,2)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      enddo

      return
      end
```

이것마저 이용한다면 메인은 정말 간결하게 다음과 같이 될것이다.

```
      program test

      real a(2,2), b(2,2), c(2,2), d(2,2), e(2,2)

      call read_array(a)
      call read_array(b)
      call read_array(c)

      call sum_array(a,b,d)
      call sum_array(c,a,e)

      call write_array(d)
      call write_array(e)

      stop
      end
```

처음에 함수와 서브루틴의 차이점에서 서브루틴의 경우에는 결과값이 어디에 저장이 되는지 명확하지 않다고 했다. 여러분들이야 이미 여러분이 직접 프로그램을 작성하고 있으니 어느 값이 입력 변수가 되고 어느 변수가 출력인지 쉽게 알 수 있지만 만약 다른 사람이나 아니면 여러분이 시간이 좀 지난 뒤에 위의 프로그램을 본다면 알 수 있을까 ? 이부분은 좀더 생각을 해볼 문제이다. 그러므로 프로그램을 작성할 때에는 이 과목의 처음부분에서 얘기했듯이 적절한 주석문을 달아서 설명을 해 줘야만된다. 배열의 합을 계산하는 부분에 주석을 달아보자.

```
      subroutine sum_array(x,y,z)
c
c 이 서브루틴은 주어진 두개의 배열의 합을 계산하는 루틴이다.
c 주어진 배열은 모두 실수형으로 선언된 변수를 사용한다.
c input  : x, y
c output : z
c
      real x(2,2), y(2,2), z(2,2)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
do i=1,2
   do j=1,2
      z(i,j)= x(i,j)+y(i,j)
   enddo
enddo

return
end
```

위와 같이 주석을 달아놔야지만 나중에 언제든지 본인 자신이던지 아니면 다른 사람이 보더래도 위의 서브루틴을 이해할 수 있을 수 있다. 꼭 주석과 들여쓰기 하는 버릇을 들이기 바란다. 또한 변수 이름도 아무런 의미 없는 것을 사용하지 말고 되도록이면 의미있는 변수를 사용해서 다시 보더래도 이해가 쉽게 만들어 둬야만된다. 사실 예전 5-6년 전만 하더래도 포트란에서 사용할 수 있는 변수이름 크기의 제약이 있었다. 8자리만의 변수를 사용하기 때문에 만약 a라는 변수가 유량을 의미하는 변수인 경우 단순히 fa 정도로만 해서 쓰는 경우도 있었지만 현재는 그러한 제약이 없으므로 flowrate_a로 변수를 만들어 쓰는 것이 훨씬 이해하기 쉽다.

마지막으로 부프로그램들을 이용할 때 장점에 대해서 얘기를 해보자. 위에서 봤듯이 단순히 메인 프로그램이 간결해 지는 것 외에 다른 장점은 없을까 ? 1000줄 짜리 프로그램을 작성하는 것보다 부프로그램을 이용해서 300줄을 바꿨다는 것만이 장점일까 ? 아니다. 잘 생각해 보자. 여러분이 배역을 합을 계속적으로 계산하는 프로그램을 작성한다고 하자. 이러한 경우 물론 실수 없이 계속 계산하는 부분을 복사와 붙여넣기를 이용해서 정확하게 하였다면 물론 문제가 없을 수 있다. 그렇지만 1000줄짜리 프로그램을 하루만에 작성한다는 것은 상당한 무리가 따를꺼고 그렇다면 며칠에 나눠서 프로그램을 작성할 수 밖에 없을 것이다. 그런 경우 하다가 보니 중간에 프로그램을 잘못 카피해 와서 문제가 생기는 경우가 발생할 수도 있을 것이다. 이러한 것을 부프로그램들은 피할 수 있게 해준다. 부프로그램으로 프로그램을 작성하는 경우에는 반복되는 프로그램에서 여러분의 실수를 최대한 줄여주고 작성을 완료한 다음에 디버깅을 하는 경우에도 단순히 부프로그램 부분만을 체크해주면 되기때문에 디버긴 시간을 줄여준다. 물론 기가막힌 프로그래밍 실력을 갖추고 있어서 한번 프로그램을 짜면 머릿속에서 자연히 컴파일러가 작동을 해서 오류를 찾아내면서 진행을 한다면 문제가 없겠지만 필자의 경우에는 프로그램을 작성하는 시간보다 어떤 경우에는 프로그램의 오류를 잡는데 더 많은 시간을 보냈고 또 이러한 문법적 오류를 잡는데 보낸 시간보다 정말 논리적 오류를 잡는데 더 많은 시간을 허비해 왔다. 물론 여러분이 이러한 일이 생기지 않을꺼라고 얘기할 수 있을지 모르겠지만 하여간 대부분의 경우에는 부프로그램으로 프로그램을 조각조각 만들면 나중에 다른 프로그램을 작성하는 경우에도 이전에 만든 프로그램을 가져다가 쓰고 그리고 컴파일에 걸리는 시간도 줄일 수 있다. 이부분에 대한 것은 다음 장에서 얘기를 하겠다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

B.4.3 부프로그램 컴파일

이번 장에서 위에서 작성한 부프로그램들을 컴파일하고 링크하는 방법에 대해서 얘기를 해 보자. 지금까지 여러분들이 해온 방법은 단순하게 하나의 파일에 모든 프로그램들을 넣고 그 하나의 파일만을 컴파일해서 사용해 왔다. 그런 경우 생기는 문제점과 해결 방법에 대해서 논의를 해 보자.

일단 더하기와 빼기를 서브루틴으로 작성을 하고 곱하는기는 함수로서 작성을 한다음 하나의 파일에 저장을 하자.

```
program test

integer i, j, k

i=2
j=3

call iconst_minus(i,j,k)

write(*,*) k

call iconst_plus(i,j,k)

write(*,*) k

k=iconst_multi(i,j)

write(*,*) k

stop
end

subroutine iconst_minus(i1,j1,k1)

k1=i1-j1

return
end

subroutine iconst_plus(i1,j1,k1)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      k1=i1+j1

      return
      end

      integer function iconst_multi(i1,j1)

      iconst_multi=i1*j1

      return
      end
```

이렇게 하나의 파일에 저장을 한다음 간단히 `df test1.f`라고 입력을 하면 실행 파일을 만들 수 있다. 그러면 만약 이렇게 해 두고 난 후에 iconst_minus라는 서브루틴을 수정해야 되는 경우에는 수정을 하고 난 후 다시 `df test1.f`를 실행해야 된다. 사실 이부분에 대해서 얘기를 해야되지만 일단 넘어가고 이 파일을 나눠서 4개의 파일을 바꾼 다음 어떻게 해야되는지 생각해 보자. 각자의 이름에 따라 파일을 저장하면 된다.

```
      program test

      integer i, j, k

      i=2
      j=3

      call iconst_minus(i,j,k)

      write(*,*) k

      call iconst_plus(i,j,k)

      write(*,*) k

      k=iconst_multi(i,j)

      write(*,*) k

      stop
      end
---------------------------------------------
      subroutine iconst_minus(i1,j1,k1)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      k1=i1-j1

      return
      end
------------------------------------------------
      subroutine iconst_plus(i1,j1,k1)

      k1=i1+j1

      return
      end
------------------------------------------------
      integer function iconst_multi(i1,j1)

      iconst_multi=i1*j1

      return
      end
```

이 프로그램의 각각의 쏘스 코드를 test.for, iconst_minus.for, iconst_plus.for, iconst_multi.for라고 저장을 했다면 이 파일들을 어떻게 하면 실행 파일로 만들 수 있을까 ? 간단히 다음과 같이 하면 된다.

```
df test.for iconst_minus.for iconst_plus.for iconst_multi.for
```

이렇게 하면 포트란 컴파일러는 일단 main 프로그램을 찾는다. 찾아보면 test.for가 main이라는 것을 알고 나머지는 단순한 부 프로그램이라는 것을 파악한 후 실행파일을 만든다. 그런데 예전에 사용하던 포트란 컴파일러의 경우에는 메일 프로그램의 이름으로 실행파일을 만들었으나 명령창에서 작업을 하는 경우 현재 필자가 사용하고 있는 포트란 컴파일러의 경우(Visual Fortran 6.0)에는 `df` 명령어 다음에 바로 나와있는 쏘스 코드의 이름으로 실행 파일을 만든다. 위의 경우에는 `test.exe`을 만드나 다른 파일이 있는 경우에는 그 쏘스코드의 이름으로 바꿔버린다. 그러나 통합개발 환경을 이용해서 프로젝트를 만들어서 작업하는 경우에는 만들고자 했던 프로젝트의 이름으로 실행파일을 만든다. 그리고 나머지 디렉토리에 남아 있는 파일들은 없다. 원래는 컴파일 했을 때 사용된 obj 파일들이 남아 있어야 되나 실행파일을 만들고 난 후에는 특별히 유지하지 않고 지워버린다. `df`라는 명령어는 컴파일과 링크를 동시에 해주는 프로그램이다.

자 여기서 잘 생각을 해보자. 실행 파일을 만들기위해서는 두가지 단계를 거쳐야 된다.

- 쏘스 파일을 목적코드(object code)로 만드는 단계

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

- object code와 다른 필요한 라이브러리들을 이용해 실행 파일을 만드는 단계

이렇게 두단계를 거쳐야 되는데 여러개의 부프로그램으로 구성된 프로그램을 하나의 파일로 작성을 하는 경우에는 이 바뀐 부분을 적용하기위해서는 파일을 컴파일해야되고 또한 만들어진 object code를 링크해서 실행파일을 만들어야 된다. 그런데 그렇게 하다가 보면 컴파일하는 시간이 증가되게 된다. 물론 여러분들이 작성하는 프로그램은 그렇게 심하게 차이가 나지 않는다. 그렇지만 만약 쏘스 코드의 파일이 100kb가 되는 경우에는 얘기가 틀려진다. 현재 만들고 있는 강의록에서 이 포트란에 관련된 파일의 크기가 60kb가 넘는다. 그렇지만 줄은 3000라인 가까이 된다. 만약 이렇게 만들어진 포트란 쏘스 파일을 컴파일 하는데는 상당한 시간이 소요된다. 물론 모든 것은 컴퓨터의 사양에 따라 변하겠지만 그래도 그렇게 하는 것은 정말 바보스러운 일이다. 그러면 여러개의 쏘스 파일로 나눈 경우에 위와 같이 컴파일해서 남은 파일이 다 지워지면 만약 쏘스 파일의 한두개가 변경되는 경우 똑같이 모든 파일을 컴파일 해야된다. 그러므로 `df` 명령어를 그냥 사용하지 말고 포트란에 관한 강의의 처음 부분에 있는 `/compile_only`라는 옵션을 사용해서 일단 모두 컴파일을 한 후 링크를 해서 사용해야만 한다.

```
df /compile_only test.for iconst_minus.for iconst_plus.for iconst_mul
```

이렇게 해도 되고 아니면 모든 파일을 다 치는 것이 귀찮으면 다음과 같이 해도 된다.

```
df /compile_only *.for
```

이렇게 하면 모든 쏘스 코드에 대한 컴파일을 시행해서 obj 파일들을 생성해 준다. 그러면 이러한 파일을 이용해서 실행 코드를 만들기 위해서는

```
df *.obj
```

이렇게 하는 경우 먼저 이름이 오는 것이 실행파일로 만들어지므로 다음과 같이 하면 이름을 명시해 줄 수 있다.

```
df /exe:test *.obj
```

그러면 `test.exe` 파일을 만들어준다. 그러면 이렇게 사용할 부프로그램에 대한 obj코드를 만든 후에 test.for 파일이 수정이 되는 경우에는 간단히

```
df test.for *.obj
```

라고 입력을 하면 부프로그램을 다시 컴파일 하는 일 없이 그냥 test.for 파일만을 컴파일해서 obj code를 만들고 링크해서 실행 파일을 만들어준다. 그런데 만약 부프로그램 가운데 하나

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

가 변형이 되는 경우에는 어떻게 해야될까? 물론 이것하나만 컴파일 하고 나머지는 그냥 obj code를 사용하면 된다. 그렇지만 이렇게 하는 경우 좀 번거러워진다. 왜냐면 이미 이 파일의 obj code가 있기 때문에 위와 같이 wildcard(* 기호) 사용해서는 원하는 결과를 얻을 수 없다. 그렇다면 해당되는 수정된 파일의 obj code를 지우고 나서야 사용할 수 있을 것이다. 이러한 불편 때문에 make라는 유틸리티가 있다. 이 부분은 나중에 설명을 하겠다. 일단 지금은 단순하게 부프로그램으로 구성된 프로그램의 컴파일 방법에 대해서만 배우자.

B.4.4 라이브러리 만들기

전장에서 부프로그램으로 구성이 된 쏘스 프로그램을 컴파일 하는 방법에 대해서 배웠다. 그럼 이제 한번 더 생각을 해보자. 만약 내가 원하는 여러가지 부 프로그램을 만들었는데 현재는 이것을 변형할 필요를 전혀 못 느끼고 있다. 그런데 이러한 부프로그램으로 구성된 것을 사용하기 위해서는 항상 여러개의 쏘스 코드를 가지고 다녀야된다. 좀 불편하지 않은가? 그래서 나온것이 라이브러리다. 라이브러리는 여러가지 부프로그램의 모음집이다. 도서관에 가면 여러가지 책이 있고 이 가운데 내가 필요로 하는 책만 뽑아서 보면 되듯이 라이브러리는 여러가지 부 프로그램으로 구성이 되어 있고 이 가운데 내가 필요로하는 프로그램만을 뽑아서 현재 사용하고자 하는 프로그램의 링크에 사용하면 되는 것이다. 일단 이번 장에서 사용하는 예제는 위에서 만든 것을 이용해보자.

라이브러리를 만들고 유지보수하기 위해서는 `lib`이라는 명령어를 사용한다. lib이라는 프로그램의 옵션은 다음과 같다.

```
F:\work\test1>lib
Microsoft (R) Library Manager Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

usage: LIB [options] [files]

   options:

      /CONVERT
      /DEBUGTYPE:CV
      /DEF[:filename]
      /EXPORT:symbol
      /EXTRACT:membername
      /INCLUDE:symbol
      /LIBPATH:dir
      /LINK50COMPAT
      /LIST[:filename]
      /MACHINE:{ALPHA|ARM|IX86|MIPS|MIPS16|MIPSR41XX|PPC|SH3|SH4}
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
        /NAME:filename
        /NODEFAULTLIB[:library]
        /NOLOGO
        /OUT:filename
        /REMOVE:membername
        /SUBSYSTEM:{NATIVE|WINDOWS|CONSOLE|WINDOWSCE|POSIX}[,#[.##]]
        /VERBOSE

F:\work\test1>
```

여기서 다른 것보다 여러분이 알아야 될 옵션은 라이브러리에 파일을 더하거나 빼거나 아니면 동일한 이름의 새로만든 파일로 교체하는 일이다. 일단 제일 간단하게 라이브러리를 만들어보자.

```
lib /out:test.lib *.obj
```

이렇게 입력을 하면 현재 디렉토리에 있는 파일을 가운데 obj 파일들만을 모아서 test.lib이라는 파일로 만들어준다. 그런데 이렇게 하면 이 파일에 무엇이 들어 있는지 모른다. 그러면 다른 옵션을 을 이용해서 현재 사용하고자 하는 라이브러리에 어떠한 파일들이 들어 있는지 알아보자.

```
lib /list:test.lst test.lib
```

여기서 '.lst' 라는 확장자를 사용한다고 정해진 것은 아니지만 이렇게 정해두면 나중에 봐도 무슨 파일인지 알 수 있다. 요즘은 확장자의 숫자에 제한이 없으니까 ' .list '라고 만들어도 될꺼다. 그렇지만 필자는 거의 10년전부터 포트란을 사용해 왔기 때문에 아직 버릇이 이렇게 그냥 3자리로 확장자를 만든다. 하여간 이 파일에는 현재 라이브러리에 들어 있는 파일에 대한 정보를 가지고 있다. 파일을 내용을 보면

```
F:\work\test1>more test.lst
iconst_minus.obj
iconst_multi.obj
iconst_plus.obj
```

아주 간단히 좀전에 넣었던 파일들이 들어 있다고 되어 있다. 여러분이 직접 라이브러리를 만드는 경우에는 꼭 이렇게 list 파일을 만들어서 같이 보관하기 바란다. 그래야지 나중에 봐도 무슨 파일이 이 라이브러리 안에 들어 있는지 알 수 있다.

그러면 이번에는 라이브러리의 유지 관리적인 측면을 고려해 보자. 일단 라이브러리에서 하나의 파일을 빼보자.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
lib /remove:iconst_plus.obj test.lib
```

이렇게 입력을 하면 test.lib에서 iconst_plus 루틴을 빼버린다. 그러면 이제 다시 존재하고 있는 라이브러리에 루틴을 하나 더해보자.

```
lib /out:test1.lib iconst_plus.obj test.lib
```

위와 같이 하면 빠진 obj code를 다시 넣어주게 된다. 이것은 물론 다른 obj code를 넣더래도 다 가능하다. 그러면 또 하나 위의 예제에는 라이브러리에서 특정 obj code를 없애버렸는데 그렇게 하지 말고 obj code를 빼내어 보자.

```
lib /extract:iconst_plus.obj test1.lib
```

이렇게 하면 원하는 obj code를 라이브러리에서 추축할 수 있다. 그러면 기존에 들어 있는 코드를 업데이트하기 위해서는 어떻게 해야 될까 ? 그건 단순히 다시 더해주면 된다. 그러면 있던 코드와 이름이 동일하면 교체를 해 버린다.

```
F:\work\test1>more test.lst
iconst_minus.obj
iconst_multi.obj
iconst_plus.obj
F:\work\test1>lib /out:test.lib iconst_minus.obj test.lib
```

위와 같이 해주면 기존에 들어있는 obj code를 업데이트 해준다.

지금까지는 라이브러리를 만드는 방법에 대해서만 논의를 했다. 그럼 이제는 이 라이브러리를 이용한 컴파일 방법에 대해서 생각을 해보자. 가장 간단한 방법은 그냥 라이브러리를 컴파일 줄때 적당한 옵션으로 같이 컴파일을 해 주면 된다.

```
df test.for /link test.lib
```

이렇게 해 주면 다 끝이다. 아주 간단히 그냥 라이브러리의 이름만 명시를 해 주고 `link`라는 옵션만 명시를 해 주면 된다. 이렇게 실행 파일을 만드는 경우에는 단순히 메인 프로그램만 컴파일 하면 되므로 시간이 절약되고 또한 윗장에서 설명을 한 것과 같이 디버깅을 하는 경우에도 단순히 현재 작업을 하고 있는 메인 프로그램만을 검사하고 라이브러리로 만든 원본 파일을 한번만 검사해 보면 된다. 그리고 라이브러리의 위치에 대해서 알아보자. 라이브러리는 어디에 있어도 되는 것이 아니라 현재 작업을 하고 있는 디렉토리에 있던가 아니면 시스템에서 정해둔 곳에 있어야 된다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
F:\work\test1>echo %LIB%
F:\Program Files\Microsoft Visual Studio\DF98\IMSL\LIB;
F:\Program Files\Microsoft Visual Studio\DF98\LIB;
F:\Program Files\Microsoft Visual Studio\VC98\LIB
```

윈도우 시스템에서는 시스템이 정해둔 변수에 대한 값을 알기위해서는 echo %VarName%을 해주면 된다. 그러므로 현재 시스템에서 정해둔 라이브러리 디렉토리를 알기 위해서는 위와 같은 명령어를 사용하면 된다. 이곳 가운데 한 곳에 넣어두면 어디서든지 내가 원하는 라이브러리를 사용할 수 있다. 그렇지 않은 경우에는 작업하는 디렉토리에 같이 들어 있어야만 된다.

B.4.5 EXTERNAL 문 사용하기

IMSL을 이용하기 전에 external이라는 특별한 선언문에 대해 일단 알아야 된다. 부 프로그램을 만드는 경우 각각의 인수에 대한 선언이 필요하게 되고 그 인수는 상수값이 될 수도 있고 배열 값이 될 수도 있다는 것은 이미 배운 내용이다. 그런데 만약 함수를 넘겨줘야 되는 경우에는 어떻게 프로그램을 작성해야 될까 ? 이 부분에 대한 문제를 다룰 수 있는 것이 external 문이다. 이 external 문을 잘못 이해하는 경우에는 나중에 상당히 힘들어지므로 차근차근 잘 이해를 해 보자.

```
      program test

      real a, b

      a=2
      b=f(a)

      write(*,*) a, b

      stop
      end
c---------------------------------------------
      real function f(x)

      f=x**2

      return
      end
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

위의 예제는 제곱값을 구해주는 함수를 작성한 뒤 이 함수에 적당한 값을 대입해서 값을 계산하는 루틴이다. 이렇게 프로그램을 작성하는 경우에는 위와 같이 프로그램을 작성한 후 컴파일하면 문제는 해결이 된다. 그러면 위의 함수값에 대한 도함수 값을 계산하는 서브루틴을 추가해 보자. 이 서브루틴은 단순하게 도함수 값을 계산하고자 하는 값과 원하는 함수만이 주어지면 된다.

```
      program test

      real a, b

      a=2

      b=(f(a+0.0001) - f(a))/0.0001

      write(*,*) a, b

      stop
      end
c-----------------------------------------------
      real function f(x)

      f=x**2

      return
      end
```

이 프로그램의 실행 결과는 다음과 같다.

```
C:\WORK>test
   2.000000       3.995895

C:\WORK>
```

원래는 x^2의 도함수는 $2x$이므로 $x = 2$ 일때의 값은 4가 나와야 되나 계산결과는 4에 근접한 값을 얻었지 정확히 4인 값은 아니다. 자 그러면 이런 경우를 생각해 보자. 함수만 주어진다면 그 함수의 원하는 값에서 도함수 값은 어떻게 계산을 해야될까 ? 아마도 위의 프로그램을 적당히 수정을 하면 되지 않을까 ? 그러면 이렇게 하면 될 것이다.

```
      real function deriv(f,x)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      real del

      del=0.0001

      deriv=(f(x+del) - f(x))/del

      return
      end
```

이러한 함수를 만들면 원하는 함수와 값을 계산하고자 하는 값만을 대입을 하면 계산할 수 있을 것이다. 그럼 위의 함수를 이용해 도함수 값 계산 프로그램을 만들어보자.

```
      program test

      real a, b

      a=2
      b=deriv(f,a)

      write(*,*) 'derivative value of f at', a,'is ', b

      stop
      end
c---------------------------------------------
      real function f(x)

      f=x**2

      return
      end
c---------------------------------------------
      real function deriv(f,x)

      real del

      del=0.0001

      deriv=(f(x+del) - f(x))/del

      return
      end
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

자 이 프로그램이 이상이 있을지 생각해 보자. 눈으로 보기에는 전혀 이상이 없다. 그렇다면 한번 컴파일을 하고 실행을 해 보자. 아마도 다음과 같은 메시지를 보게 될 것이다.

```
C:\WORK>test
forrtl: severe (157): Program Exception - access violation
Image              PC        Routine      Line      Source
test2.exe          0043B4B1  Unknown      Unknown   Unknown
test2.exe          0040103D  Unknown      Unknown   Unknown
test2.exe          0042CE29  Unknown      Unknown   Unknown
test2.exe          00424064  Unknown      Unknown   Unknown
KERNEL32.dll       77E57903  Unknown      Unknown   Unknown
```

컴파일을 하는 단계까지는 문제가 없었지만 실행하는 단계에서 문제가 생긴다. 이러한 것이 run-time error이다. 뭔가 문제가 생겼다. 그러면 여러분은 어떠한 결론을 얻을 것인가 ? 프로그램 작성이 잘못되었거나 그런 것은 분명히 아니다. 그랬으면 이미 컴파일 하는 단계에서 문제가 생겼을테니.. 그럼 뭘까 ? 그건 deriv 함수에 f라는 함수의 이름을 인수로 사용했기 때문에 생긴 문제이다. 이 deriv 함수는 f라는 인수값이 함수의 값인지 모르기 때문에 일단 컴파일 단계에서는 문제를 발생하지 않았지만 실제 실행을 하는 단계에서 문제가 있다는 것을 깨닮은 것이다. 그러므로 이러한 문제를 해결하기 위해서 external 이라는 키워드가 필요한 것이다. external 이라는 키워드는 현재 부프로그램에서 사용하는 인수 가운데 어떠한 것은 함수라는 것을 알려주는 역할을 한다. 그러므로 위의 프로그램을 다음과 같이 수정을 해야된다.

```
      program test

      real a, b

      external f

      a=2
      b=deriv(f,a)

      write(*,*) 'derivative value of f at', a,'is ', b

      stop
      end
c----------------------------------------------
      real function f(x)

      f=x**2
```


ⓒcopyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      return
      end
c---------------------------------------------
      real function deriv(f,x)

      real del

      del=0.0001

      deriv=(f(x+del) - f(x))/del

      return
      end
```

위와 같이 수정을 한 후 실행을 하면 다음과 같은 원하는 결과값을 얻을 수 있다.

```
C:\WORK>test
 derivative value of f at   2.000000    is    3.995895
```

잘 기억을 해야된다. 만약 함수를 부프로그램에서 사용하기 위해서는 꼭 그 사용하자하는 것을 external로 선언을 해서 사용해야만 된다. 꼭 기억하자. 이 부분이 함수를 설명하는 부분에 있어야 될꺼 같은데 그 부분에서 설명을 하면 왜 필요한지 모르기 때문에 이렇게 부프로그램을 작성하고 사용하는 부분에서 설명하게 되었다. 그러면 이러한 것을 바탕으로 IMSL을 사용하는 방법에 대해서 알아보자.

B.4.6 IMSL 사용하기

IMSL(International Mathematical and Statistical Library)는 다양한 포트란 부프로그램의 모음집이다. 크게 두부분으로 나워진다.

- MATH/LIBRARY general applied mathematics and special functions
- STAT/LIBRARY statistics

이렇게 두부분으로 나워진 부프로그램들은 사용하는 정확도에 따라 또 두 부분으로 나눠진다.

- Single precision : function itself or S or A prefix
- Double precision : D prefix

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

이렇게 구성이 된 IMSL은 실제 다음과 같이 구분이 되어서 라이브러리로 저장이 되어 있다.

- SMATHS : Single-precision MATH library, one of the IMSL FORTRAN 77 Numerical Libraries.
- SMATHD : Double-precision MATH library, one of the IMSL FORTRAN 77 Numerical Libraries.
- SSTATS : Single-precision STAT library, one of the IMSL FORTRAN 77 Numerical Libraries.
- SSTATD : Double-precision STAT library, one of the IMSL FORTRAN 77 Numerical Libraries.
- SF90MP : Fortran 90 MP library, a new generation of Fortran 90-based algorithms, optimized for multiprocessor and other high-performance systems.

위의 각각의 라이브러리를 필요에 따라 사용하면 된다. 위의 라이브러리들 가운데 MATH 라이브러리에 있는 루틴들은 다음과 같다.

- Linear Systems
- Eigensystem Analysis
- Interpolation and Approximation
- Integration and Differentiation
- Differential Equations
- Transforms
- Nonlinear Equations
- Optimization
- Basic Matrix/Vector Operations
- Utilities

또한 위에서 Basic Matrix/Vector Operations와 Utilities들은 말그래도 기본적으로 사용하는 벡터 행렬과 관련된 루틴들과 간단한 유틸리티들이므로 실제 수치해석에서 사용할 것은 아니다. 그 외의 것들은 수치해석에서 직접적으로 사용할 루틴들이다.

이중에서 Nonlinear Equations에 관련된 것들을 한번 보자. 다음과 같이 크게 세 경우에 대해서 여러가지 루틴들이 존재하고 있다. 일단 이 세 경우에서 구하고자 하는 함수의 형태에

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

따라서 함수가 다항식으로 표시가 되는지 아니면 다항식 외의 다른 형태의 함수로서 표시되는지에 따라 크게 나눠진다. 일반적인 경우에는 다항식 보다는 그냥 함수의 형태로 표시될 것이다. 그럼 그냥 함수로서 표시가 되는 경우에도 근이 허근이 있는지 없는지에 따라 달라질 것이고 그에 따라 다른 루틴들이 존재한다. 마지막으로 시스템의 근을 찾는 경우가 있는 이 부분은 교과서 6.5에서 nonlinear equation의 시스템 부분에서 설명을 약간 했었다. 이러한 문제는 주로 고차의 미분방정식에서 많이 만들어지는데 그러한 문제를 해결하기 위해서는 시스템으로 구성된 식의 근을 찾는 루틴을 이용해서 해결을 해야만 된다.

- Zeros of a Polynomial
  - Real coefficients using Laguerre method ZPLRC
  - Real coefficients using Jenkins-Traub method ZPORC
  - Complex coefficients ZPOCC
- Zero(s) of a Function
  - Zeros of a complex analytic function ZANLY
  - Zero of a real function with sign changes ZBREN
  - Zeros of a real function ZREAL
- Root of a System of Equations
  - Finite-difference Jacobian NEQNF
  - Analytic Jacobian NEQNJ
  - Broyden's update and Finite-difference Jacobian NEQBF
  - Broyden's update and Analytic Jacobian NEQBJ

위와 같이 동일한 문제에 여러가지 루틴이 동시에 존재할 수 있으므로 여러분들은 적당한 루틴을 골라서 사용해야만 한다. 이러한 모든 IMSL 루틴에 대한 설명을 다 할 수는 없고 필요에 따라 찾아서 사용하면 된다. 그 부분은 HELP 파일을 참조하기 바란다. 그럼 일단 이 IMSL 라이브러리를 이용해서 프로그램을 작성하는 방법에 대해서 익히도록하자. 예로서 여러분이 가장 처음 만나게될 ZREAL이라는 루틴에 대해서 알아보자.

```
ZREAL/DZREAL (Single/Double precision)

Find the real zeros of a real function using Muller’s method.

Usage
CALL ZREAL (F, ERRABS, ERRREL, EPS, ETA, NROOT, ITMAX, XGUESS, X, INFO)
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
Arguments
F . User-supplied FUNCTION to compute the value of the function of which
    a zero will be found. The form is F(X), where
    X . The point at which the function is evaluated. (Input)
        X should not be changed by F.
    F . The computed function value at the point X. (Output)
        F must be declared EXTERNAL in the calling program.

ERRABS . First stopping criterion. (Input)
    A zero X(I) is accepted if ABS(F(X(I)).LT. ERRABS.

ERRREL . Second stopping criterion is the relative error. (Input)
    A zero X(I) is accepted if the relative change of two successive
    approximations to X(I) is less than ERRREL.

EPS . See ETA. (Input)

ETA . Spread criteria for multiple zeros. (Input)
    If the zero X(I) has been computed and ABS(X(I) - X(J)).LT.EPS, where
    X(J) is a previously computed zero, then the computation is restarted
    with a guess equal to X(I) + ETA.

NROOT . The number of zeros to be found by ZREAL. (Input)

ITMAX . The maximum allowable number of iterations per zero. (Input)

XGUESS . A vector of length NROOT. (Input)
    XGUESS contains the initial guesses for the zeros.
X . A vector of length NROOT. (Output)
    X contains the computed zeros.
INFO . An integer vector of length NROOT. (Output)
    INFO(J) contains the number of iterations used in finding the J-th ze
    when convergence was achieved. If convergence was not obtained in ITM
    iterations, INFO(J) will be greater than ITMAX.
```

아마도 여러분이 보는 형식은 위의 형태보다는 좀더 보기 좋은 형태로 되어 있을 것이다. 하여간 중요한 것을 보자. 일단 이 ZREAL이라는 부프로그램은 Muller 방식에 의해 함수의 근을 구하는데 사용되는 루틴으로 Usage 부분에 보면 어떻게 불러야되는지 나와있다. 상당히 복잡하다고 느낄꺼지만 차근차근보자. 이러한 부프로그램을 보면 가장 먼저해야될 일은 어떠한 값이 입력치로 넣어야 될 것이며 어떠한 값이 출력치로 나올지를 결정해야만 된다. 그럼 입력변수가운데 가장 먼저 사용하는 인수가 F라는 함수이다. 이 함수는 external로 선언이

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

되어 있어야 되면 찾고자 하는 함수를 나타내야만 된다. 두번째 인수는 함수가 그 근에서는 0에 가까워진다. 그러므로 이렇게 근이 된다면 언제 중단해야될지를 결정해 주는 부분이다. 세번째 인수는 계속적으로 근으로 추정되는 값이 구해질때 두 찾아진 근을 언제 올바른 근으로서 받아드릴지를 결정하는 부분이다. 네번째와 다섯번째는 연속적으로 찾아진 두 근이 EPS 안에 있다면 나중에 찾아진 근에 ETA만큼의 값을 더해서 다시 찾는다. 다음의 인수는 찾고자 하는 근의 갯수를 의미하며 ITMAX는 계속해서 근을 찾을 수 없기 때문에 얼마나 계산을 하고 그만둘지를 결정해 준다. 마지막 입력 값은 XGUESS로 근의 근사치를 넣어주면 된다. 사실 이 부분은 정확히 알 수 없으므로 대강 정해주면 계산을 해 준다. 이제 출력 변수에 대한 것을 보자. 출력변수는 별다른 것이 없이 원하는 근과 그 근에서의 함수 값 등을 알면된다. F라는 변수는 입력으로서도 사용이 되었지만 출력변수의 값으로도 사용이 되었다. 즉, 입력시에는 근을 구하고자 하는 함수의 값이지만 근을 찾고난 후 출력을 할 경우에는 출력함수의 값이 된다. 그리고 근에 대한 정보를 가지고 있는 X라는 변수가 있고 마지막으로 계산이 제대로 되었는지 아니면 문제가 없는지에 관한 INFO라는 변수 값이 있다. 계산이 잘 되는 경우에는 문제가 없지만 잘되지 않는 경우에는 이 변수의 값을 잘 살펴봐야만 된다.

자 그럼 이제 예제 이용해서 ZREAL 루틴을 사용해 보자. 일단은 간단한 이차방정식의 근을 찾아보자. 원하는 식은 다음과 같다.

$$f(x) = x^2 + 2x - 6 \tag{B.1}$$

이 부분을 포트란으로 구성을 하면

```
real function f(x)
real x

f = x*x + 2.0*x - 6.0
return
end
```

그 다음 ZREAL을 이용하는 메인 프로그램을 구성하자.

```
program test

parameter  (nroot=2)
integer    info(nroot)
real       x(nroot), xguess(nroot)
external   f

data xguess/4.6, -193.3/

eps    = 1.0e-5
```


©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr

```
      errabs = 1.0e-5
      errrel = 1.0e-5
      eta    = 1.0e-2
      itmax  = 100

      call zreal (f, errabs, errrel, eps, eta, nroot, itmax, xguess,
     &            x, info)

      do i=1,nroot
         write(*,*) x(i), info(i), f(x(i))
      enddo

      stop
      end
```

위의 프로그램을 살펴보자. 일단 윗부분에서는 사용할 배열에 대한 선언이 있다. 그 다음 f라는 함수를 external로 선언을 해서 사용하겠다고 선언하였다. 이 부분은 이미 전 섹션에서 설명한 내용이다. 그 다음 초기값에 대한 것을 DATA를 이용해서 선언하였다. 그 다음 필요한 데이터 값을 설정하고 난 후 ZREAL 부프로그램을 불렀다. 문제 해결에 성공을 했다면 별 무리없이 결과가 나올 것이다. 그 다음은 출력으로 임의로 결정을 해주면 된다. 출력값으로 근인 X 배열변수와 INFO 배열변수가 넘어온다고 했으니 그 값과 그때 함수의 값을 출력해 보았다. 이 프로그램을 컴파일하는 것에 대해 알아보자. 위에서 부프로그램 컴파일과 라이브러리를 이용한 컴파일 방법에 대한 부분을 잘 봤다면 어려움 없이 할 수 있다. 그리고 마지막으로 이번에 사용할 IMSL 라이브러리의 이름은 smaths.lib이다. 위의 프로그램에서는 double precision에 해당하는 부프로그램을 사용하지 않았으므로 smaths.lib이면 충분하다.

```
J:\work\numerical\test>df test.for f.for /link smaths.lib
```

이렇게 해서 얻는 실행파일을 실행하면 다음과 같은 결과를 얻을 수 있다.

```
J:\work\numerical\test>test
   1.645751               4  2.4348299E-07
  -3.645751               5 -3.8731326E-07
```

INFO 변수에는 근을 찾는데 성공했다면 계산에 필요했던 계산과정의 합이 저장이 된다. 각각 4번과 5번만에 계산 결과를 찾을 수 있었다. 그리고 그때 함수의 값은 보이는 것과 같이 거의 0에 가까운 값이다.

자 이것으로 IMSL 라이브러리를 사용하는 방법까지 모든 설명이 끝났다. 아마도 이것으로 appendix를 마무리질려고 하는데 사실 어떠한 것들이 더 필요할지는 모르겠다. 좀더 여러분들이 하는 것을 지켜보고 추가할 사항이 있는 경우 더 추가를 해서 충실한 강의록이 되도록 노력하겠다.

©copyright 2001-2004 Taechul Lee: tclee@prosys.korea.ac.kr