Fundamental Numerical Analysis E

(1) Nov. 11, 2004

Lecturer: Ying CHEN (陳 迎)
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng (鄭 波)
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
Today

• Overview of Course

• Training on ECC System

• Distribute ECC Account

• Getting started
  Login, Logout, basic Unix, …
Objectives & Schedule

1-2: Introduction
  Guidance to ECC, Basic Unix operation

2-3: Review of C language

4-10: Numerical Algorithm
  - Numerical integration and derivatives
  - Linear algebraic equations
  - Root finding, Nonlinear sets of equation
  - Ordinary differential equations
  - Monte Carlo method
Course Calendar

Dec. 2: No lesson
Course Assessment

• In-class/take-home exercises: 50%

• Take-home final assignment: 50%

• Email to us
  ying@q.t.u-tokyo.ac.jp
  zheng@zzz.t.u-tokyo.ac.jp
References


• Unix tutorial for beginners: http://www.ee.surrey.ac.uk/Teaching/Unix/
Course Information

http://gwp01.t.u-tokyo.ac.jp/kouryu/students/classes/ica.htm

- Announcement
- Lecture Notes (will be uploaded in 1-2 days after lecture)
- Take-home Exercises
- Final assignment
Requirement

- Be on time
- Don’t talk each other loudly in lesson
- Home work/Assignment must be turned in by the due date
- Study after class
- Please ask questions freely in lecture
Important Rules

- Your account is only for yourself, only for study and research (Don’t abuse!) If you miss the password, need to get a new one (ECC office).

- No food and beverage in terminal room.

- Make sure your terminal “Shut down” before leaving.

- Take care Limitations (Dir: 500MB, Mail: 100MB)
  https://secure.ecc.u-tokyo.ac.jp/quota.html
Guidance to ECC System

Education Computer Center (ECC) System

http://www.ecc.u-tokyo.ac.jp/

FNA-E (1), Nov. 11, 2004
Changing Password

https://secure.ecc.u-tokyo.ac.jp/cgi-bin/passwd.cgi

User’s name
Current password
New password
New password
“Three kinds of world”

iMac

Unix

Windows
1. iMac
2. Happy Hacking keyboard
3. Mouse
iMac: Login/Logout

Login

Logout

Shutdown

User Name

Password

Return

logout+shut down
iMac: Applications

- Finder
- Mail
- Safari: Web browser
- X11, Term: Unix
- Remote Desktop Connection: Windows environment
- Microsoft: Word, Excel, PowerPoint
- Acrobat 6.0
- Photoshop Elements 2.0
- Mathematica 5.0
- STATA

...more...
Unix Environment

- C, C++
- Fortran, Fortran90
- Pascal
- Java

Application

- X11 or Terminal
- Open Unix

Exit
Close
Windows Environment

**login**

Remote Desktop Connection

User Name

Password

**Application**

- GiveWin2
- Microsoft Office
- Mozilla
- Opera
- SAS (statistics)
- TSP (statistics)
- Adobe Acrobat 6.0 Standard
- Adobe Photoshop Element
- video LAN
...

**logout**

Windows NT のログアウト

OK

キャンセル
File Sharing Between Unix and iMac

iMac/Finder

Unix Home Directory → read Unix files

Unix/ Terminal

directory: /Desktop → read iMac files
Mail Service

Mail address: LoginName@mail.ecc.u-tokyo.ac.jp

Mail client

Web Mail: https://wm.ecc.u-tokyo.ac.jp/

Mail (need setting)
Terminal Rooms

Asano Campus
• Information Technology Center 1F 5F

Komaba Campus
• Information education Center 1F-4F
• Komaba Library 2F

Hongo Campus
• Main Library 2F, 3F
• International Student Center
• School of Eng. Bldg.1, 2F
• Law, Bldg.1, 1F
...
Fundamental Numerical Analysis E

(2) Nov. 18, 2004

Lecturer: Ying CHEN (陳 迎)
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng (鄭 波)
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
Today

- Basic Unix commands

Now

Please login to your Unix ...
Unix: Operating System

of workstation and multi-users server
Unix: File System, Path

```
/  root
   /usr, /bin,…
   /
   bin   etc   usr   lib   dev
   /
usr2   /
   bin   lib
   /
   chen   taka
   /
file1   test   file1
   /
   file2
```

```
/usr/usr2/chen, …
```

```
/usr/usr2/chen/test, ~/test
```
Unix Environment (X11)

Shell for Command:

> cd test

prompt line command
>, $, #, ...
Basic Unix Commands

Try ..., What happens?

> date

> pwd

> history
Format of Unix command:

```
Command -Option Variables
```

Option: followed by “-”, extended functions
Variables: file names, path, dir., ...

eg. `ls -l hello`
    `mv -i file1 file2`
Basic Unix Commands

Try More ... list contents of directory

> ls

    tying@as301> ls
    GNUstep       atokdicts.tar  registry
    GNUstep.sun   mytest       test

> ls -F

    tying@as301> ls -F
    GNUstep/     atokdicts.tar  registry/
    GNUstep.sun/ mytest       test/

> ls -l

> ls -a
Basic Unix Commands

Making / Deleting directory

```bash
> mkdir test11.18

| tying@as301> mkdir test 11.18 |
| tying@as301> |
| tying@as301> ls |
GNUstep        atokdicts.tar  registry |
GNUstep.sun    mytest          test   |
| tying@as301> ls -F |
| GNUstep/      atokdicts.tar  registry/ |
| GNUstep.sun/  mytest          test/   |
| test 11.18    |

> rmdir dir_name
```
Basic Unix Commands

Change directory

> cd test11.18

  tying@as301> cd test 11.18
  tying@as301>
  tying@as301> pwd
  /home/tying/test 11.18

> cd ..  Go one directory up hierarchy

  tying@as301> cd ..
  tying@as301> pwd
  /home/tying

> cd  Return to home directory
Practice 1

1) Make directory "/practice1" under ~/test11.18;

2) Confirm "/practice1" has been created successfully;

3) Go back to ~/test11.18,
   Go back to home directory.

4) Go to "/Desktop" to see what it is?
Basic Unix Commands

Editing a file: Emacs

> emacs
> emacs &
> emacs testfile &

different?

Command: 2 ways

- menu on top
- line command in bottom

Save file: c-x c-s
Quit emacs: c-x c-c
Open file: c-x c-f
......
Basic Unix Commands

Displaying a file

> less file_name  ("q": quit display)
> more file_name
> cat file_name

Deleting a file

> rm file_name
> rm -i file_name
Basic Unix Commands

Copying a file

> cp file_original file_new

Renaming a file/directory
Move a file/directory

> mv file_old file_new
1) Make another directory inside “/test11.18” called “/practice2”;

2) Make a file “hello” inside “/test11.18/practice2” (including your first name inside the “hello”);

3) Display the content of file “hello”;

3) Use “ls”, “pwd”, “cd” to explore your file system, Find the full pathname of your file “hello”.
Getting Help: Online Manual

> man command

> man ls
> man cp
> man mv
...

Basic Unix Commands
Simple Unix Commands

Combined commands: Realize complicated functions

**Redirect:** >  
(Write output of command into a file)

```bash
ls -l > filelist
less filelist > wc filelist
```

**Pipe:** |  
(Output of 1st command as input of 2nd command)

```bash
ls -l | wc
```
Comparing difference of 2 files

```bash
> diff file1 file2
```

Extracting lines containing specific characters

```bash
> grep "characters" file
```

eg. ```bash
> grep your_name hello
```
“Tricks”

- Copy and Paste by mouse key
- Re-use command, several ways:
  - !!
  - >history
    - > !#command in history list
  - up-arrow
- wildcard (meta-character): *
Fundamental Numerical Analysis E

(4) Dec. 9, 2004

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
Today

Continue: Review of Basic C language
Start: Numerical Algorithm

Now

Please login to your Unix ...
4. Function

Basic Structure of program

File.c

external variables

main ( )

{variables statements}

external variables

function1 (…)

{local variables function body}

function2 (…)

{local variables function body}

Variable Scope
/* finding root of */
x**3-5x**2+16x-82=0 */
#include <stdio.h>
#include <math.h>
#define TINY 0.00001

float f(float x) {
  ...
}

float xpoint(float x1,float x2) {
  ...
}

float root(float x1,float x2) {
  ...
}

main {
  ...
}
5. Input and Output

Character level I/O

getchar(): read one char. from stdin.
putchar(): write one char. to stdout.

High level I/O (formatted I/O)

scanf(): read data from stdin.
printf(): formatted output to stdout.

I/O to and from files

fscanf(): read data from a file.
fprintf(): formatted output to a file.
#include <math.h>
#include<stdio.h>
#define TINY 0.00001

main() {
    float f(float x);
    float xpoint(float x1,float x2);
    float root(float x1,float x2);
    float x1,x2,f1,f2,x;
    
    FILE * fp;
    if((fp=fopen("out","w"))==NULL)
        {printf("cannot open the file\n");
         exit(0); }

    do {
        printf("input x1,x2: \n");
        scanf("%f,%f",&x1,&x2);
        f1=f(x1);
        f2=f(x2);
    }while(f1*f2>=0);
    x=root(x1,x2);
    fprintf(fp,"a root of eq. is :%8.5f\n",x);
    fclose(fp); }
...
5. Preprocessor

**File inclusion:**

`#include <filename>`

`#include “filename”`

**Head file:**

`#include <filename.h>`

`#include <stdio.h>`

`#include <math.h>`

`#include <string.h>`

One program includes more than one source file: in `main()`:

`#includes “file1.c”`

`#includes “file2.c”` ...
Practice 4-4  Several source files

main( ) → pract44.c
float f(float x) → file1.c
float xpoint(float x1, float x2) → file2.c
float root(float x1, float x2) → file3.c

2 ways to Run the program

inside main():
#include “file1.c”
#include “file2.c”
#include “file3.c”
Compile:
>gcc -lm pract44.c
>gcc -lm pract44.c file1.c
file2.c file3.c
double acos(double x) -- Compute arc cosine of x.
double asin(double x) -- Compute arc sine of x.
double atan(double x) -- Compute arc tangent of x.
double atan2(double y, double x) -- Compute arc tangent of y/x.
double ceil(double x) -- Get smallest integral value that exceeds x.
double cos(double x) -- Compute cosine of angle in radians.
double cosh(double x) -- Compute the hyperbolic cosine of x.
div_t div(int number, int denom) -- Divide one integer by another.
double exp(double x) -- Compute exponential of x
double fabs (double x) -- Compute absolute value of x.
double floor(double x) -- Get largest integral value less than x.
double frexp(double x, int *expptr) -- Breaks down x into mantissa and exponent of no.
labs(long n) -- Find absolute value of long integer n.
double ldexp(double x, int exp) -- Reconstructs x out of mantissa and exponent of two.
div_t ldiv(long number, long denom) -- Divide one long integer by another.
double log(double x) -- Compute log(x).
double log10 (double x ) -- Compute log to the base 10 of x.
double modf(double x, double *intptr) -- Breaks x into fractional and integer parts.
double pow (double x, double y) -- Compute x raised to the power y.
double sin(double x) -- Compute sine of angle in radians.
double sinh(double x) -- Compute the hyperbolic sine of x.
double sqrt(double x) -- Compute the square root of x.
void srand(unsigned seed) -- Set a new seed for the random number generator (rand).
double tan(double x) -- Compute tangent of angle in radians.
double tanh(double x) -- Compute the hyperbolic tangent of x.

Compile:
>gcc -lm file.c
5. Preprocessor

Macro substitution:

#define Name replacement_text

eg. #define PI 3.1415927
#define ACCURACY 0.0000001

Conditional compile

Conditional inclusion
1. Introduction:
   Error, Accuracy and Stability

2. Root Finding
Basic Idea

Solve a mathematical problem only by repeating a set of simple operations.
1. Introduction

Roundoff Error: Characteristic of hardware

A number in integer representation is exact. Floating-point representation in arithmetic is not exact.

\[ s \times M \times B^{e-E} \]

Where, 
- \( s \): a sign bit (interpreted as plus or minus);
- \( e \): an exact integer exponent;
- \( M \): exact positive integer mantissa;
- \( B \): base of the representation (usually \( B = 2 \));
- \( E \): exponent bias, fix integer for given machine.

Any arithmetic operation among floating numbers should be thought of as introducing an additional fractional error of at least “machine accuracy” - roundoff error.

\[ \frac{1}{3} = 0.333333333333333333333333 \]
1. Introduction

Truncation Error: Characteristic of Software

Numerical algorithms compute “discrete” approximations to some desired “continuous” quantity.

The discrepancy between the true answer and the answer obtained in a practical calculation is called the truncation error.
Some Principles

1) Subtraction of 2 similar number loses much accuracy;
2) Small divisor leads large error in quotient;
3) Different order of operations influences the accuracy of data;
   (some number will be “eaten”)
4) Try to decrease the times of arithmetic operations.
5) Use long integral and double floating.
Error developing in calculation: Algorithm Stability

Unstable method: The roundoff error becomes “mixed into” the calculation at an early stage is successively magnified until it comes to swamp the true answer.

eg.

\[ I_n = \int_0^1 \frac{x^n}{x+5} \, dx \quad n = 0,1,2,...,20 \]

2 recursion relations:

\[ I_n + 5I_{n-1} = \frac{1}{n} \quad I_0 \approx 0.182322 \quad \rightarrow \quad I_{20} = 0.423988 \times 10^8 \rightarrow \text{Unstable!} \]

\[ I_{n-1} = \frac{1}{5n} - \frac{1}{5} I_n \quad I_{20} \approx 8.73016 \times 10^{-3} \quad \rightarrow \quad I_0 = 0.182322 \rightarrow \text{Stable!} \]
2. Roots Finding

Root(s) of $f(x)$:

Solution(s) of 1D-equation
$f(x) = 0$

Graphically, $x$-intercepts of the curve of $f(x)$
2. Roots Finding

Two steps in finding roots

1) Setting Initial value or initial interval; Scanning for rough distribution of roots;

2) Refine the rough value of roots.
2. Roots Finding

Plot Tool: GNUplot

Gnuplot is a command-line driven program for producing 2D and 3D plots.

In Unix: Start

```bash
%gnuplot
enter the "gnuplot" mode:
>plot sin(x)
...
>quit
```

http://www.cs.uni.edu/Help/gnuplot/, ...
**Practice 5**

**GNUplot: Try...**

In Unix:

Start GNUplot mode:

%gnuplot

Inside GNUplot:

> plot sin(x)
> plot x*x
> plot sin(x), x*x
> plot sin(x) w d
> plot sin(x) w l
> plot sin(x) w i
> set xrange [-5:5]
> plot sin(x) - x*x
> ...  
> splot x*x + sin(y)
> replot - x*x + y*y
> quit
**Intermediate Value Theorem**, assuming 2 values: $x_1$ and $x_2$

$f(x_1) < 0$ and $f(x_2) > 0$,

If this is the case (and the function $f$ is continuous), there must be at least one value $x_0$ that falls between $x_1$ and $x_2$.

**Bisection method cannot fail.**
2. Roots Finding

Secant and False Position

Secant method:
Extrapolation through the two most recently evaluated points.

\[ x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \]

False position method:
Interpolation lines (dashed) through the most recent points that bracket the root.

\[ x \rightarrow x_1 \text{ if } f(x)f(x_1) > 0 \]
\[ \text{else } x \rightarrow x_2 \]
Newton’s method extrapolates the local derivative to find the next estimate of the root.

Generally, it is powerful and converges quadratically.

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]
Practice 6  Newton-Raphson Method

To write a program to find one root of function:

\[ 5.0 \sin(x) = x \quad (0, 4) \]

Accuracy=0.000001.

```
tying@ux101> a.out
input x1: 
2
 1  2.82658434  2.82658434
 2  2.60457015  2.60457015
 3  2.59575796  2.59575796
 4  2.59573913  2.59573913
 5  2.59573913  2.59573913
a root of eq. is :  2.59573913
```
// Newton-Raphson, of 5*sin(x)=x (0,4)
#include <math.h>
#include<stdio.h>
define TINY 0.000001

void function(float x, float*py, float*pyy)
{
    *py=5.0*sin(x)-x;
    *pyy=5.0*cos(x)-1.0;
}

float root(float a1) {
    float a,f,ff;
    int i=0;
Practice 6  Newton-Raphson Method

do {
    function(a1,&f,&ff);
    a=a1-f/ff;
    a1=a;
    i++;  
    printf("%3d%14.8f%14.8f\ n",i,a1,a);
}  while(fabs(f)>=\NY);
return(a1);
}

main() {  
    float x1,x;
    printf("input x1: \ n");
    scanf("%f",&x1);
    x=root(x1);
    printf("a root of eq. is :%14.8f\ n",x);
}
Exercise 1 (Submit by Dec. 22)

Find all roots of following function in [-5,5]:

\[10 \sin(x) - x = 0\]

1) Write a program to do pre-search roots;
2) To find all roots in [10,10] by using Bisection, Secant, False Position, Newton-Raphson method, respectively, at accuracy = 0.0000001;
3) Take one root as example, comparing the convergent speed of 4 method. Summarize the results on to a WORD file.
Request on Exercises

- Subject of the email should include message like:
  “FNA-exe1”, “FNA-exe2”, ...
- Subject or Body should include
  “Student Number” if you don’t use ECC account.
- You should Run the program before sending.

Email to both of us:

ying@q.tu-tokyo.ac.jp
zheng@zzz.tu-tokyo.ac.jp
Fundamental Numerical Analysis E

(5) Dec. 16, 2004

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
## Content of Numerical Algorithm

<table>
<thead>
<tr>
<th>1. Introduction:</th>
<th>(4) 12/9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error, Accuracy and Stability</td>
<td></td>
</tr>
<tr>
<td>2. Roots Finding</td>
<td></td>
</tr>
<tr>
<td>3. Interpolation and extrapolation</td>
<td>(5) 12/16</td>
</tr>
</tbody>
</table>

**Now**

Please login to your Unix ...
Problem Setting

Given a table of values of a function in \([a,b]\):

\[ f(x_i), \ i=1, \ldots, n \]

Estimate:

\[ f(x) \] for arbitrary \( x \)

- Graphically: drawing a smooth curve through points
- Different from fitting
3. Interpolation and Extrapolation

General Idea

• 2-stage method:
  (1) Fit an interpolating function to the data points provided,
  (2) Evaluate that interpolating function at the target point $x$.

Not the best way:
  - Inefficient
  - roundoff error

• Most practical schemes:
  Start at a nearby point $f(x_i)$, then add a sequence of (hopefully) decreasing corrections as information from other $f(x_i)$’s incorporated.
3. Interpolation and Extrapolation

Order of interpolation: Number of tabulated points used.

- Increasing order does not always lead to higher precision.
- Extrapolation is prone to error.

A smooth function (solid line) is more accurately interpolated by a high-order polynomial.

A function with sharp corners or rapidly changing higher derivatives is less accurate by a high-order polynomial.
3. Interpolation and Extrapolation: Overview

Interpolation

- Aitken Interpolation
- B-Spline
- Berlekamp-Massey Algorithm
- Bézier Curve
- Bicubic Spline
- Bulirsch-Stoer Algorithm
- C-Determinant
- Cardinal Function
- Chebyshev Approximation Form

- Cubic Spline
- Gauss's Interpolation Formula
- Hermite's Interpolating Polynomials
- Internal Knot
- Interpolation
- Lagrangian Coefficient
- Lagrange Interpolating Polynomials
- Lebesgue Constants
- Moving Average
- Muller's Method
- Neville's Algorithm
- Newton's Divided Differences
- NURBS Curve
- NURBS Surface
- Richardson Extrapolation
- Spline
- Thiele's Interpolation Formula
- Thin Plate Spline

Lagrange formula: first published by Waring in 1779, rediscovered by Euler in 1783, and published by Lagrange in 1795.
3. Interpolation and Extrapolation: Overview

- **Polynomial**
  \[ P(x) = \sum_{j=1}^{n} P_j(x), \quad P_j(x) = y_j \prod_{k=1 \atop k \neq j}^{n} \frac{x - x_k}{x_j - x_k}. \]
  Neville’s Algorithm

- **Rational function**
  \[ R_{i(i+1)\ldots(i+m)} = \frac{P_{\mu}(x)}{Q_{\nu}(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu} \]
  Bulirsch-Stoer’s Algorithm

- **Spline**
  - **B-Spline (Bezier Curve):**
    \[ C(t) = \sum_{i=0}^{n} P_i B_{i,n}(t). \]
  - **Cubic Spline:**
    \[ y = Ay_j + By_{j+1} + Cy_j'' + Dy_{j+1}'' \]

- **High dimension**
3. Interpolation and Extrapolation: Polynomial

Lagrange Formula

An \( n-1 \) degree polynomial passed through the \( n \) points in \([a, b]\):

\[
y_1 = f(x_1), \quad y_2 = f(x_2), \ldots, \quad y_n = f(x_n)
\]

\[
P(x) = \sum_{j=1}^{n} P_j(x), \quad P_j(x) = y_j \prod_{k=1, k \neq j}^{n} \frac{x - x_k}{x_j - x_k}.
\]

\[
P(x) = \frac{(x - x_2)(x - x_3)...(x - x_N)}{(x_1 - x_2)(x_1 - x_3)...(x_1 - x_N)} y_1 + \frac{(x - x_1)(x - x_3)...(x - x_N)}{(x_2 - x_1)(x_2 - x_3)...(x_2 - x_N)} y_2
\]

\[+ \cdots + \frac{(x - x_1)(x - x_2)...(x - x_{N-1})}{(x_N - x_1)(x_N - x_2)...(x_N - x_{N-1})} y_N \sim x^{n-1}
\]
3. Interpolation and Extrapolation

\( n=2, \text{ Linear interpolation} \)

\[
\frac{x_2 - x_1}{y_2 - y_1} = \frac{x - x_1}{y - y_1}
\]

\[
P(x) = y = \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2
\]

\( n=3, \text{ Parabola interpolation} \)

\[
P(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3
\]
3. Interpolation and Extrapolation: Polynomial

Neville’s Algorithm

\[ P_{i(i+1)...(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)...(i+m-1)} + (x_i - x)P_{(i+1)(i+2)...(i+m)}}{x_i - x_{i+m}} \]

\[ C_{m,i} \equiv P_{i...(i+m)} - P_{i...(i+m-1)} \]
\[ D_{m,i} \equiv P_{i...(i+m)} - P_{(i+1)...(i+m)} \]

\[ D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \]
\[ C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \]
Demonstrate the Lagrange interpolation by using $\sin(x)$.

Input number of known points $N$:

Tabulated data: $x, \sin(x_i), i=1,...,n$

$Estimate \sin(x) \text{ at } x = (-0.05 + i/10) \times \pi$

Program: 3 functions
- define a flexible array: vector/freevector
- Lagrange interpolation: polint
- value of interpolating point: main
/** pract7 lagrange interpolation, Dec. 16, 2004 */
#include <stdio.h>
#include <math.h>
// #include <malloc.h> in our current system, unnecessary
#define PI 3.1415926

float* vector(long n)  // make a flexible dimension array
{
    /* allocate n floats */
    float* pv;
    pv = (float*)malloc((size_t)((n + 1) * sizeof(float)));
    if(!pv)
        printf("allocation failure in vector()\n");
    return pv;
}

void freevector(float* pv)
{
    free(pv);
}
void polint(float xa[], float ya[], int n, float x, float* y, float* dy)
{
        int i, m, ns = 1;
        float den, dif, dift, ho, hp, w;
        float*c,*d;

        dif = fabs(x - xa[1]);
        c = vector(n);
        d = vector(n);
        for(i=1; i<=n; i++)
        {  //find index ns as closest table point
            if((dift = fabs(x - xa[i])) < dif)
            {  
                ns = i;
                dif = dift;
            }
            c[i] = ya[i];  //initial c,d
            d[i] = ya[i];
        }
        *y = ya[ns];  //initial approxi. to y. same “ya[ns-1]”
        ns=ns-1;
}

Practice 7  Polynomial Interpolation
for(m=1; m<n; m++) { // for each column of tableau
    for(i=1; i<= n-m; i++) { // loop over current c, d
        ho = xa[i] - x;
        hp = xa[i+m] - x;
        w = c[i+1] - d[i];
        if((den = ho - hp) == 0.0)
            printf("Error in routine point\n");
        den = w / den;
        d[i] = hp * den; // update c, d
        c[i] = ho * den;
    }
    if(2*ns <(n-m)) // decide which correction, c or d?
        *dy = c[ns + 1];
    else {
        *dy = d[ns]; ns=ns-1; } // same as “d[ns-]”
    *y +=*dy;
}
freevector(c);
freevector(d);
int main(void) {
    int i, n;
    float dy, f, x, y, *xa, *ya;

    printf("generation of interpolation tables\n");
    printf(" ... sin(x) 0<x<\pi\n");
    printf("how many entries go in these tables?\n");
    if (scanf("%d", &n) == EOF) return 1;
    xa = vector(n);
    ya = vector(n);

    printf("\nsine function from 0 to \pi\n");
    printf("tabulated points: n =\n"); // input n
    printf("ni %9s %13s \n", "x0", "f(x0)" );
    for (i = 1; i <= n; i++) {
        xa[i] = i * PI / n; // known points
        ya[i] = sin(xa[i]);
        printf("%d %12.6f %12.6f\n", i, xa[i], ya[i]);
    }
}
# Practice 7  Polynomial Interpolation

```c
printf("n%9s %13s %16s %13s\n", "x", "f(x)", "interpolated", "error");
for(i=1; i<=10; i++) {
    x = (-0.05 + i / 10.0) * PI;  // interpolating points
    f = sin(x);
    polint(xa, ya, n, x, &y, &dy);
    printf("%12.6f %12.6f %12.6f %4s %11f\n", x, f, y, " ", dy);
}
freevector(xa);
freevector(ya);
return 0;
}```
**Practice 7  Polynomial Interpolation**

RUN:

tying@as301> a.out
generation of interpolation tables
... \( \sin(x) \)  \(0 < x < \pi\)
how many entries go in these tables?
3
sine function from 0 to \( \pi \)

<table>
<thead>
<tr>
<th>tabulated points: ( n = 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>( f(x) )</th>
<th>interpolated</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.157080</td>
<td>0.156434</td>
<td>0.185113</td>
<td>-0.680912</td>
<td></td>
</tr>
<tr>
<td>0.471239</td>
<td>0.453990</td>
<td>0.496882</td>
<td>-0.369143</td>
<td></td>
</tr>
<tr>
<td>0.785398</td>
<td>0.707107</td>
<td>0.730709</td>
<td>-0.135316</td>
<td></td>
</tr>
<tr>
<td>1.099557</td>
<td>0.891007</td>
<td>0.964536</td>
<td>0.020568</td>
<td></td>
</tr>
<tr>
<td>1.413717</td>
<td>0.987688</td>
<td>0.987688</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>1.727876</td>
<td>0.987688</td>
<td>0.987688</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>2.042035</td>
<td>0.891007</td>
<td>0.886594</td>
<td>0.098510</td>
<td></td>
</tr>
<tr>
<td>2.356194</td>
<td>0.707107</td>
<td>0.730709</td>
<td>0.098510</td>
<td></td>
</tr>
<tr>
<td>2.670354</td>
<td>0.453991</td>
<td>0.496882</td>
<td>0.107171</td>
<td></td>
</tr>
<tr>
<td>2.984513</td>
<td>0.156434</td>
<td>0.185113</td>
<td>0.055209</td>
<td></td>
</tr>
</tbody>
</table>

Save data into files

data-in
data-out

To see result by GNUplot:

\( \text{plot "data-in"} \)
\( \text{plot "data-out" using 1:2} \)
\( \text{plot "data-out" using 1:3} \)
\( \text{plot "data-in" w p, "data-out" u 1:3 w l} \)

...
3. Interpolation and Extrapolation: **Spline**

**Cubic Spline**

A piecewise polynomial function that can have a locally very simple form, yet at the same time be globally flexible and smooth.

2 requires: \((x_i, x_{i+1})\) smooth in the first derivative; continuous in the second derivative.

\[
y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1}
\]

\(~X\) \(\sim X^3\)

\[
A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \quad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j} \quad C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \quad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2
\]

\((y'')\) is 2\(^{nd}\) derivative of \(y\)

\[\text{...eq.(1)}\]
3. Interpolation and Extrapolation:  Spline

\( y'(x_i) \) continuity \( \rightarrow \)

\( n-2 \) linear equations in the \( N \) unknowns \( y''_i, i = 2, \ldots, n-1 \).

\[
\frac{x_j - x_{j-1}}{6} y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3} y''_j + \frac{x_{j+1} - x_j}{6} y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}
\]

Boundary condition \( \rightarrow \) 2 equations

- **Natural cubic spline:** \( y''_1 = 0, y''_2 = 0 \)
- **Specified 1st derivative:** \( y'_1, y'_2 \)

\( \rightarrow \)  All values of \( y''(x_i), \ i = 1, \ldots, n \)

2 steps: 1) Calculate 2\textsuperscript{nd} derivative: “spline”
2) Calculate value of interpolating point: “splint”
Practice 8  Cubic Spline Interpolation

/* pract8 cubic spline interpolation, Dec. 16, 2004 */
#include <stdio.h>
#include <math.h>
// #include <malloc.h> in our current system, unnecessary
#define PI 3.1415926

float* vector(long n)
{
    /* allocate n floats */
    float* pv;
    pv = (float*)malloc((size_t)((n + 1) * sizeof(float)));
    if(!pv)
        printf("allocation failure in vector()");
    return pv;
}

void freevector(float* pv)
{
    free(pv);
}
```c
void spline(float x[], float y[], int n, float yp1, float ypn, float y2[])
{
    int i,k;
    float p,qn,sig,un,*u;
    u=vector(n-1); // following setting boundary of i=1
    if (yp1 > 0.99e30) // if y'(1)=\infty, set natural boundary cond.
        y2[1]=u[1]=0.0;
    else {
        y2[1] = -0.5; // or, set to specified y'(1)
        u[1]=(3.0/(x[2]-x[1]))*((y[2]-y[1])/(x[2]-x[1])-yp1);
    }

    // following solve eq.(2)
    for (i=2;i<=n-1;i++) { // tridiagonal algorithm: decomposition
        sig=(x[i]-x[i-1])/ (x[i+1]-x[i-1]);
        p=sig*y2[i-1]+2.0;
        y2[i]=(sig-1.0)/p;
        u[i]=(y[i+1]-y[i])/ (x[i+1]-x[i]) - (y[i]-y[i-1])/ (x[i]-x[i-1]);
        u[i]=(6.0*u[i]/ (x[i+1]-x[i-1])-sig*u[i-1])/p;
    }
}```
// following setting boundary of i=n
if (ypn > 0.99e30)    // set another boundary “natural” or “value”
    qn=un=0.0;
else {
    qn=0.5;
    un=(3.0/ (x[n]-x[n-1]))*(ypn-(y[n]-y[n-1])/ (x[n]-x[n-1]));
}
y2[n]=(un-qn*u[n-1])/ (qn*y2[n-1]+1.0);
for (k=n-1;k>=1;k--) // tridiagonal algorithm: backsubstitution
    y2[k]=y2[k]*y2[k+1]+u[k];
freevector(u);

void splint(float xa[], float ya[], float y2a[], int n, float x, float *y)
{
    int klo,khi,k;
    float h,b,a;
Practice 8  Cubic Spline Interpolation

```c
klo=1;
khi=n;
while (khi-klo > 1) { // by mean of bisection to find right interval
    k=(khi+klo) >> 1; // same as "k=(khi+klo)/2;"
    if (xa[k] > x) khi=k;
    else klo=k;
}
// now, khi and klo bracket the input "x"

h=xa[khi]-xa[klo];
if (h == 0.0) printf("Error in routine polint\n");
a=(xa[khi]-x)/h;
b=(x-xa[klo])/h;

*y=a*ya[klo]+b*ya[khi]+((a*a*a-a)*y2a[klo]+(b*b*b-b)*y2a[khi]) *(h*h)/6.0;
}```
int main(void) {
    int i, nfunc;
    float f, x, y, yp1, ypn, *xa, *ya, *y2;

    xa = vector(NP);
    ya = vector(NP);
    y2 = vector(NP);

    printf("Tabulated points N= %d", NP);
    printf(" \nsine function from 0 to pi \n");
    for (i = 1; i <= NP; i++) {
        xa[i] = i * PI / NP;
        ya[i] = sin(xa[i]);
        printf("%d  %12.6f  %12.6f \n", i, xa[i], ya[i]);
    }
    yp1 = cos(xa[1]);
    ypn = cos(xa[NP]);
spline(xa,ya,NP,yp1,ypn,y2); // Call spline to get second derivatives

printf("\n%9s %13s %17s %8s\n","x","f(x)","interpolation","error");

for (i=1;i<=10;i++) {
    x=(-0.05+i/10.0)*PI;
    f=sin(x);
    splint(xa,ya,y2,NP,x,&y); // Call splint for interpolations
    printf("%12.6f %12.6f %12.6f %12.6f\n",x,f,y, f-y);
}

freevector(xa);
freevector(ya);
freevector(y2);
return 0;
RUN:
tying@as301> a.out
Tabulated points N= 5
sine function from 0 to pi
1  0.628319  0.587785
2  1.256637  0.951057
3  1.884956  0.951057
4  2.513274  0.587785
5  3.141593  0.000000

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
<th>interpolation</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.157080</td>
<td>0.156434</td>
<td>0.148925</td>
<td>0.007510</td>
</tr>
<tr>
<td>0.471239</td>
<td>0.453990</td>
<td>0.453542</td>
<td>0.000449</td>
</tr>
<tr>
<td>0.785398</td>
<td>0.707107</td>
<td>0.706941</td>
<td>0.000166</td>
</tr>
<tr>
<td>1.099557</td>
<td>0.891007</td>
<td>0.890860</td>
<td>0.000147</td>
</tr>
<tr>
<td>1.413717</td>
<td>0.987688</td>
<td>0.987413</td>
<td>0.000275</td>
</tr>
<tr>
<td>1.727876</td>
<td>0.987688</td>
<td>0.987425</td>
<td>0.000264</td>
</tr>
<tr>
<td>2.042035</td>
<td>0.891007</td>
<td>0.890814</td>
<td>0.000192</td>
</tr>
<tr>
<td>2.356194</td>
<td>0.707107</td>
<td>0.706850</td>
<td>0.000256</td>
</tr>
<tr>
<td>2.670354</td>
<td>0.453991</td>
<td>0.453999</td>
<td>-0.000008</td>
</tr>
<tr>
<td>2.984513</td>
<td>0.156434</td>
<td>0.156400</td>
<td>0.000035</td>
</tr>
</tbody>
</table>
3. Interpolation and Extrapolation

Comparison of Polynomial and Cubic Spline

Global polynomial function: Changes drastically

Local piecewise polynomial function: smooth

http://www.wam.umd.edu/~petersd/interp.html
Exercise 2  (Submit by Jan. 12, 2005)

There is a 10 points data set:

\[
\begin{array}{ccccccccccc}
  x: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
  y: & 1 & 8 & 27 & 64 & 125 & 216 & 343 & 512 & 729 & 1000 \\
\end{array}
\]

Calculate the \textit{y-value} at \(x = 2.5, 5.7\) and \(9.75\) by using cubic spline interpolation method.

Marry Christmas and Happy New Year!

See you on Jan. 13, 2005!
Request on Exercises

- **Subject of the email should include message like:**
  
  “FNA-exe1”, “FNA-exe2”, ...

- **Subject or Body should include**
  
  “**Student Number**” if you don’t use ECC account

- **You should Run the program before sending.**

**Email to both of us:**

ying@q.t.u-tokyo.ac.jp
zheng@zzz.t.u-tokyo.ac.jp
Fundamental Numerical Analysis E

(6) Jan. 13, 2005

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
Content of Numerical Algorithm

1. Introduction: Error, Accuracy and Stability
2. Roots Finding
3. Interpolation and extrapolation
4. Numerical Integration

Now
Please login to your Unix ...
/* pract8 cubic spline interpolation, Dec. 16, 2004 */
#include <stdio.h>
#include <math.h>
// #include <malloc.h> in our current system, unnecessary
#define PI 3.1415926
#define NP 10

float* vector(long n)
{
    /* allocate n floats */
    float* pv;
    pv = (float*)malloc((size_t)((n + 1) * sizeof(float)));
    if(!pv)
        printf("allocation failure in vector()\n");
    return pv;
}

void freevector(float* pv)
{
    free(pv);
}
4. Numerical Integration

- Perhaps with the **longest history** in numerical algorithms.

- Useful as integrals of elementary functions **cannot** always be computed **analytically**.

- Surprisingly **simple** when written as a computer program.

\[
I = \int_a^b f(x) \, dx \quad h = \frac{b - a}{n}
\]
4. Numerical Integration

Summary of numerical integral methods

- Newton-Cotes Formulas (single interval formulas)

  the most commonly used numerical integration methods, approximate the integration by replacing the function with polynomials. *(can be used in discrete data sets)*

\[ \int_{x_1}^{x_2} f(x)dx = h \left[ \frac{1}{2} f_1 + \frac{1}{2} f_2 \right] + O(h^3 f''') \]

\( n=1, \) Trapezoidal rule

\[ \int_{x_1}^{x_3} f(x)dx = h \left[ \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{1}{3} f_3 \right] + O(h^5 f^{(4)}) \]

\( n=2, \) Simpson’s (1/3) rule

\[ \int_{x_1}^{x_4} f(x)dx = h \left[ \frac{3}{8} f_1 + \frac{9}{8} f_2 + \frac{9}{8} f_3 + \frac{3}{8} f_4 \right] + O(h^5 f^{(4)}) \]

\( n=3, \) Simpson’s 3/8 rule

\[ \int_{x_1}^{x_5} f(x)dx = h \left[ \frac{14}{45} f_1 + \frac{64}{45} f_2 + \frac{24}{45} f_3 + \frac{64}{45} f_4 + \frac{14}{45} f_5 \right] + O(h^7 f^{(6)}) \]

\( n=4, \) Bode’s rule
4. Numerical Integration

\[ n=2, \quad \text{Linear interpolation} \]

\[
P_2(x) = \frac{x-x_2}{x_1-x_2} f_1 + \frac{x-x_1}{x_2-x_1} f_2
\]

\[
= \frac{x-x_1-h}{-h} f_1 + \frac{x-x_1}{h} f_2
\]

\[
= \frac{x}{h} (f_2 - f_1) + \left(f_1 + \frac{x_1}{h} f_1 - \frac{x_1}{h} f_2 \right)
\]

\[
\int_{x_1}^{x_2} f(x) \, dx = \int_{x_1}^{x_1+h} P_2(x) \, dx
\]

\[
= \frac{1}{2h} (f_2 - f_1) [x^2]_{x_1}^{x_2} + \left(f_1 + \frac{x_1}{h} f_1 - \frac{x_1}{h} f_2 \right) [x]_{x_1}^{x_2}
\]

\[
= \frac{1}{2h} (f_2 - f_1) (x_2 + x_1)(x_2 - x_1) + (x_2 - x_1) \left(f_1 + \frac{x_1}{h} f_1 - \frac{x_1}{h} f_2 \right)
\]

\[
= \frac{1}{2} (f_2 - f_1)(2x_1 + h) + f_1 h + x_1 (f_1 - f_2)
\]

\[
= x_1 (f_2 - f_1) + \frac{1}{2} h (f_2 - f_1) + h f_1 - x_1 (f_2 - f_1)
\]

\[
= \frac{1}{2} h (f_1 + f_2) - \frac{1}{12} h^3 f''(\xi)
\]
4. Numerical Integration

**Quadrature**

**Summary of numerical integral methods**

- **Gaussian Quadrature**
  
  provide the flexibility of choosing not only the weighting coefficients (weight factors) but also the locations (abscissas). Yield twice of accuracy as that of the Newton-Cotes formulas with the same number of function evaluations, when the function is known and smooth.

- **Improper Integral: finite, singularity**

- **Multidimensional Integral: Monte Carlo**
4. Numerical Integration  

Riemann’s Sums

Approximation: Area of strip ~ Rectangular

Left Riemann’s Sum

\[ \int_{a}^{b} f(x) \, dx = \sum_{i=1}^{n} f(x_{i-1})h \]

Right Riemann’s Sum

\[ \int_{a}^{b} f(x) \, dx = \sum_{i=1}^{n} f(x_{i})h \]

Midpoints Riemann’s Sum

\[ \int_{a}^{b} f(x) \, dx = \sum_{i=1}^{n} f\left(\frac{x_{i-1} + x_{i}}{2}\right)h \]
4. Numerical Integration

Trapezoidal Rule

**Approximation: Area of strip \text{~Trapezoidal}\)**

\[
\int_{x_1}^{x_2} f(x) \, dx = h \left[ \frac{1}{2} f_1 + \frac{1}{2} f_2 \right] + O(h^3 f'')
\]

**Composite Trapezoidal Rule**

\[
\int_{a}^{b} f(x) \, dx = \frac{h}{2} \sum_{i=1}^{n-1} \{ f(x_i) + f(x_{i-1}) \}
\]

\[
= \frac{h}{2} \left[ f(a) + 2f(x_1) + 2f(x_2) + \ldots + 2f(x_{n-1}) + f(b) \right] + O(h^2)
\]

**Error:**

\[
- \frac{(b-a) f'''(c)}{12} h^2 \sim h^2 \sim \frac{1}{N^2}
\]
4. Numerical Integration

**Trapezoidal Rule**

**Iteration Procedure**

<table>
<thead>
<tr>
<th>step</th>
<th>add $2^{N-2}$</th>
<th>points</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 1$</td>
<td>0</td>
<td>$n=2$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

(total after $N = 4$)

**Trapezoidal method** → **Romberg method**

(high order, high accuracy)
Practice 9 Trapezoidal Method

To calculate the integral by Trapezoidal method, and compare to exact value.

\[ \int_0^\frac{\pi}{2} x^2 (x^2 - 2) \sin x = ? \]

Exact result:

\[ \int_0^\frac{\pi}{2} x^2 (x^2 - 2) \sin x = 4x(x^2 - 7) \sin x - (x^4 - 14x^2 + 28) \cos x \]
/* Practice 9, trapezoidal integral, Jan. 13, 2005 */
#include <stdio.h>
#include <math.h>
#define PIO2 1.5707963
#define EPS 1.0e-7
#define JMAX 20
#define FUNC(x) ((*func)(x))

/* Test function */
double func(double x) {
    return x*x*(x*x-2.0)*sin(x);
}

/* Integral of test function- exact */
double fint(double x) {
    return 4.0*x*(x*x-7.0)*sin(x)-(pow(x,4.0)-14.0*x*x+28.0)*cos(x);
}
/* Trapezoidal */
double trapzd(double (*func)(double), double a, double b, int n) {
    double x,tnm,sum,del;
    static double s;
    int it,j;
    if (n == 1) {
        return (s=0.5*(b-a)*(func(a)+func(b)));  // 1st two points
    }else {
        for (it=1,j=1;j<n-1;j++) it*=2;
        tnm=it;  // no. of points to be added
        del=(b-a)/tnm;  // spacing of adding points
        x=a+0.5*del;
        for (sum=0.0,j=1;j<=it;j++,x+=del) sum += func(x);
        s=0.5*(s+(b-a)*sum/tnm);  // s is replaced by refined value
        return s;
    }
}
/* Main */
int main(void) {

double a=0.0,b=PI/2,s=0,os=0;
long j,nn=2;
printf("Integral of func computed with Trapezoidal: \n \n");
printf("Actual value of integral is %12.8f \n",fint(b)-fint(a));
j=1;
do {
    os=s;
    s=trapzd(func,a,b,j); // s is replaced
    if(j>=2) {nn=nn+(int)pow(2,j-2);} // no. of points
    printf("%6d %10d %20.8f \n",j,nn,s);
    j++;
} while (fabs(s-os)>=EPS); // meet accuracy or not
printf("Result is: I=%12.8f \n",s);
return 0;
}
Practice 9 Trapezoidal Method

**RUN:**
```
ing@as301> gcc -lm Pract9.c
tying@as301> a.out
```
Integral of func computed with trapezoidal:
Actual value of integral is \(-0.47915884\)
```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.90577278</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-0.02094499</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>-0.36146132</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>-0.44958376</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>-0.47175632</td>
</tr>
<tr>
<td>6</td>
<td>33</td>
<td>-0.47730767</td>
</tr>
<tr>
<td>7</td>
<td>65</td>
<td>-0.47869602</td>
</tr>
<tr>
<td>8</td>
<td>129</td>
<td>-0.47904313</td>
</tr>
<tr>
<td>9</td>
<td>257</td>
<td>-0.47912991</td>
</tr>
<tr>
<td>10</td>
<td>513</td>
<td>-0.47915161</td>
</tr>
<tr>
<td>11</td>
<td>1025</td>
<td>-0.47915703</td>
</tr>
<tr>
<td>12</td>
<td>2049</td>
<td>-0.47915839</td>
</tr>
<tr>
<td>13</td>
<td>4097</td>
<td>-0.47915873</td>
</tr>
<tr>
<td>14</td>
<td>8193</td>
<td>-0.47915881</td>
</tr>
</tbody>
</table>
```
Result is:  \(I = -0.47915881\)
4. Numerical Integration

Simpson’s Rule

Approximation: Curve of top of strip ~ Parabola

\[ \int_{x_1}^{x_3} f(x) \, dx = h \left[ \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{1}{3} f_3 \right] + O(h^5 f^{(4)}) \]

\textit{Composite Simpson’s Rule}

\[ \int_{a}^{b} f(x) \, dx = \frac{h}{3} \sum_{i=1}^{n-1} \{ f(x_{i-1}) + 4 f(x_i) + f(x_{i+1}) \} \]

\[ = \frac{h}{2} \left[ f(a) + 4 f(x_1) + 2 f(x_2) + 4 f(x_3) + 2 f(x_4) \ldots + f(b) \right] + O(h^4) \]

Error:

\[ - \frac{(b-a)f^4(c)}{180}h^4 \sim h^4 \sim \frac{1}{N^4} \]
Practice 10    Simpson’s Method

To calculate the integral by Simpson’s method, and compare to exact value.

\[
\int_{0}^{\frac{\pi}{2}} x^2 (x^2 - 2) \sin x = ?
\]

**Exact result:**

\[
\int_{0}^{\frac{\pi}{2}} x^2 (x^2 - 2) \sin x = 4x(x^2 - 7) \sin x - (x^4 - 14x^2 + 28) \cos x
\]
/* Practice 10-1, Simpson integral, Jan. 13, 2005 */
#include <stdio.h>
...

/* Test function */
double func(double x) {
    ...
}

/* Integral of test function-exact */
double fint(double x) {
    ...
}
/* Simpson-1, fixed no. of intervals */
double simpson(double (*func)(double), double a, double b, int m) {
    double xodd, xeven, sodd, seven, del;
    double sum;
    int i;
    seven = 0;
    sodd = 0;
    sum = 0;

    for (i=0; i<m; i++) {
        del = (b-a)/(2*m);
        xodd = a + del*(2*i+1);
        sodd = sodd + func(xodd);
    }
    for (i=0; i<m-1; i++) {
        del = (b-a)/(2*m);
        xeven = a + del*(2*i+2);
        seven = seven + func(xeven);
    }

    sum = del*(func(a) + 4*sodd + 2*seven + func(b))/3;
    return sum;
}
/* Main */
int main(void)
{
    double a=0.0, b=PI*2, s=0;
    int m;
    printf("Enter m (half of No. points): \n");
    scanf("%d", &m);
    printf("Integral of func computed with Simpson\n\nActual value of integral is ...\nNo. points is %d\n", 2 * m);
    printf("Result is %12.8f\n", s);
    return 0;
}

RUN:
tying@as301> a.out
Enter m (half of No. points):
50
Integral of func computed with Simpson
No. points is 100
Actual value of integral is -0.47915884
Result is -0.47915883
4. Numerical Integration

Simpson’s Rule

Compute Simpson’s integral by using Trapezoidal

\[ S = \frac{4}{3} S_{2n} - \frac{1}{3} S_n \sim O(h^4) \]
4. Numerical Integration

**Simpson’s Rule**

Approximation: Curve of top of strip ~ Parabola

\[
\int_{x_1}^{x_3} f(x) \, dx = h \left[ \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{1}{3} f_3 \right] + O(h^5 f^{(4)})
\]

**Composite Simpson’s Rule**

\[
\int_{a}^{b} f(x) \, dx = \frac{h}{3} \sum_{i=1}^{n-1} \{ f(x_{i-1}) + 4f(x_i) + f(x_{i+1}) \}
\]

\[
= \frac{h}{2} \left[ f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) \ldots + f(b) \right] + O(h^4)
\]

Error: \[
- \frac{(b - a) f^{(4)}(c)}{180} h^4 \sim h^4 \sim \frac{1}{N^4}
\]
/* Practice 10-2, Simpson integral, Jan. 13, 2005 */
#include <stdio.h>
...

/* Test function */
double func(double x)
{...}

/* Integral of test function-exact */
double fint(double x)
{...}

/* Trapezoidal */
double trapzd(double (*func)(double), double a, double b, int n)
{...}
/* Simpson-2, extended by trapezoidal */
double qsimp(double (*func)(double), double a, double b) {
    double trapzd(double (*func)(double), double a, double b, int n);
    int j;
    double s, st, ost = 0.0, os = 0.0;
    for (j = 1; j <= MAX; j++) {
        st = trapzd(func, a, b, j); // replace st by doubled intervals value
        s = (4.0 * st - ost) / 3.0; // ost is half no. interval of st's
        if (j > 5) // avoid "false" early convergent
            if (fabs(s - os) < EPS * fabs(os) ||
                (s == 0.0 && os == 0.0)) return s;
        printf("%6d  %20.8f
", j, s);
        os = s;
        ost = st;
    }
    printf("Too many steps in routine qsimp");
    return 0.0;
}
/*Main */
int main(void) {
    double a=0.0,b=PI02,s;
    printf("Integral of func computed with QSIMP\n\n");
    printf("Actual value of integral is \%14.8f\n",fint(b)-fint(a));
    s=qsimp(func,a,b);
    printf("Result is: I= \%14.8f\n",s);
    return 0;
}

RUN:
tying@as301> a.out
Integral of func computed with Simpson:

Actual value of integral is    -0.47915884
1     1.20769704
2    -0.32985091
3    -0.47496676
4    -0.47895791
5    -0.47914717
6    -0.47915813
7    -0.47915880
Result is: I=  -0.47915884
Comparing to Practice 9 Trapezoidal Method

RUN:
ying@as301> gcc -lm Pract9.c
tyting@as301> a.out
Integral of func computed with trapezoidal:
Actual value of integral is -0.47915884

|   1  |    2   |  0.90577278 |
|    2  |    3   | -0.02094499 |
|    3  |    5   | -0.36146132 |
|    4  |    9   | -0.44958376 |
|    5  |   17   | -0.47175632 |
|    6  |   33   | -0.47730767 |
|**7**|**65**|**-0.47869602**|
|    8  |  129   | -0.47904313 |
|    9  |  257   | -0.47912991 |
|   10  |  513   | -0.47915161 |
|   11  | 1025   | -0.47915703 |
|   12  | 2049   | -0.47915839 |
|   13  | 4097   | -0.47915873 |
|**14**|**8193**|**-0.47915881**|

Result is: $I = -0.47915881$
4. Numerical Integration

Comparison of different methods

- Speed
- Error

Left-hand rule
Right-hand rule
Midpoint rule
Trapezoidal rule
Simpson’s rule

http://math.furman.edu/~dcs/java/NumericalIntegration.html
Fundamental
Numerical Analysis E

(7) Jan. 20, 2005

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
## Content of Numerical Algorithm

1. **Introduction:**
   Error, Accuracy, and Stability

2. **Roots Finding**

3. **Interpolation and Extrapolation**

4. **Numerical Integration**

5. **Linear Algebraic Equations**

---

**Now**

**Please login to your Unix ...**
5. Linear Algebraic Equations

Problem Setting

A set of linear algebraic equations looks like this:

\[ a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N}x_N = b_1 \]
\[ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N}x_N = b_2 \]
\[ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3N}x_N = b_3 \]
\[ \cdots \cdots \cdots \]
\[ a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots + a_{MN}x_N = b_M \]  (1)

Here the \( N \) unknowns \( x_j, j = 1, 2, \ldots, N \) are related by \( M \) equations. The coefficients \( a_{ij} \) with \( i = 1, 2, \ldots, M \) and \( j = 1, 2, \ldots, N \) are known numbers, as are the right-hand side quantities \( b_i, i = 1, 2, \ldots, M \).
Matrix form: \[ A \cdot x = b \]

If \( N = M \), \( \det |A| = 0 \): there exists unique solution sets of \( \{x_j\} \);  
If \( N > M \), or \( N = M \) but \( \det |A| = 0 \): singular matrix, no solution, or more than one solution set.  
If \( N < M \), generally no solution.
5. Linear Algebraic Equations

Task of computational linear algebra

• Solution of the matrix equation: \( A \cdot x = b \)

• Solution of more than one matrix equation:
  \[
  A \cdot x_1 = b_1, \\
  A \cdot x_2 = b_2, \\
  \ldots
  
  \]

• Calculation of the inverse matrix \( A^{-1} \), \( A \cdot A^{-1} = A^{-1} \cdot A = I \),
  For an \( N \times N \) matrix \( A \), this is equivalent to the previous task with the unit vectors \( b_j (j = 1, 2, \ldots, N) \).

• Calculation of the determinant \( |A| \) of a square matrix \( A \).
5. Linear Algebraic Equations

Methods

- **Direct method**: widely used, also do matrix calculation, round-off error, unstable.
  - Gaussian elimination: find solutions
  - Gauss-Jordan elimination: solutions, inverse
  - LU Decomposition
  - Tridiagonal equations, Band diagonal equations, …
  - QR decomposition
  - …

- **Iterative method**: simple, convergent speed? approximation.
  - eq. (1) → \( x_i^{(k+1)} = a_{11}x_1^{(k)} + a_{22}x_2^{(k)} + \ldots + b_1 \)

- **Singularity value decomposition (SVD)**
5. Linear Algebraic Equations

Elementary operations (on rows) (keep solution set of $A$ unchanged)

1) Interchange any two rows of $A$;
2) Replace any row in $A$ by multiplying a factor to each row elements;
3) Replace any row in $A$ by a linear combination of itself and any other.
5. Linear Algebraic Eqs.  Gauss-Jordan Eliminit

Demonstration

http://www.cse.uiuc.edu/eot/modules/linear_equations/gauss_jordan/
5. Linear Algebraic Eqs. Gaussian Elimination

Elimination by column

\[
\begin{pmatrix}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4
\end{pmatrix}
\]

\[a'_{ij} = a_{ij} - a_{ik}a_{kj}/a_{kk}\]
5. Linear Algebraic Eqs. Gaussian Elimination

**Backsubstitution**

\[
\begin{pmatrix}
  a'_{11} & a'_{12} & a'_{13} & a'_{14} \\
  0 & a'_{22} & a'_{23} & a'_{24} \\
  0 & 0 & a'_{33} & a'_{34} \\
  0 & 0 & 0 & a'_{44}
\end{pmatrix}
\cdot
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4
\end{pmatrix}
=
\begin{pmatrix}
  b'_1 \\
  b'_2 \\
  b'_3 \\
  b'_4
\end{pmatrix}
\]

\[
x_4 = b'_4 / a'_{44}
\]

\[
x_3 = \frac{1}{a'_{33}} [b'_3 - x_4 a'_{34}]
\]

\[
x_i = \frac{1}{a'_{ii}} \left[ b'_i - \sum_{j=i+1}^{\nu} a'_{ij} x_j \right]
\]
5. Linear Algebraic Eqs.  Gauss-Jordan Eliminit.

Example

\[
\begin{pmatrix}
1 & 4/3 & 2/3 & 5/3 \\
0 & 1 & -1 & -5/2 \\
0 & 0 & 1 & 3 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix}
=
\begin{pmatrix}
1/3 \\
1 \\
-3/4 \\
-13/9
\end{pmatrix}
\]

\[x_4 = -13/9\]

\[x_3 = -3/4 - 3x_4\]
5. Linear Algebraic Eqs. Gaussian Elimination

**Pivoting:** Decreasing error in elimination

\[
\begin{pmatrix}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}
\]

\[d'_{ij} = d_{ij} - a_{ik} \frac{a_{kj}}{a_{kk}}\]

\[
\begin{pmatrix}
a'_{11} & a'_{12} & a'_{13} & a'_{14} \\
0 & a'_{22} & a'_{23} & a'_{24} \\
0 & 0 & a'_{33} & a'_{34} \\
0 & 0 & 0 & a'_{44}
\end{pmatrix}
\]

**Using largest element as** \(a_{kk}\)

**Full Pivoting:** interchange rows and columns.
**Partial Pivoting:** interchange rows only.
5. Linear Algebraic Eqs. Gaussian Elimination

**Tridiagonal Equations**

\[
\begin{bmatrix}
  b_1 & c_1 & 0 & \ldots \\
  a_2 & b_2 & c_2 & \ldots \\
  \vdots & \ddots & \ddots & \ddots \\
  \cdots & \cdots & a_{N-1} & b_{N-1} & c_{N-1} \\
  \cdots & \cdots & \cdots & 0 & a_N & b_N \\
\end{bmatrix}
\begin{bmatrix}
  u_1 \\
  u_2 \\
  \vdots \\
  u_{N-1} \\
  u_N \\
\end{bmatrix}
=
\begin{bmatrix}
  r_1 \\
  r_2 \\
  \vdots \\
  r_{N-1} \\
  r_N \\
\end{bmatrix}
\]

Each step of using Gaussian elimination and back substitution can be encoded very concisely.
Gauss-Jordan Elimination

- Based on Gaussian Elimination
- Calculate inverse of matrix simultaneously

\[ A \rightarrow I \rightarrow I \rightarrow B = A^{-1} \]

- No back substitution (since normalized)
**Practice 11  Gauss-Jordan Elimination**

**Perform:** gauss-Jordan elimination with full pivoting → \( \{x_j\} \)

**Comparison:** multiplication of \( A \cdot A^{-1} = I \);
values of both side of equations.

**Program:** 3 functions
- define a flexible dimensional
  array: ivector/free_ivector
  matrix: matrix/free_matrix
- gauss-Jordan elimination(also calculate \( A^{-1} \)): gaussj
- perform calculation: main
Practice 11  Gauss-Jordan Elimination

Input file: matrix1.dat

MATRICES FOR INPUT TO TEST ROUTINES
Size of matrix (NxN), Number of solutions: 3 1
Matrix A:
9.0 3.0 4.0
4.0 3.0 4.0
1.0 1.0 1.0
Solution vectors B:
7.0 8.0 3.0

NEXT PROBLEM:
Size of matrix (NxN), Number of solutions: 5 2
Matrix A:
1.4 2.1 2.1 7.4 9.6
1.6 1.5 1.1 0.7 5.0
3.8 8.0 9.6 5.4 8.8
4.6 8.2 8.4 0.4 8.0
2.6 2.9 0.1 9.6 7.7
Solution vectors B:
1.1 1.6 4.7 9.1 0.1
4.0 9.3 8.4 0.4 4.1
...
/* Pract11 Gauss-Jordan Elimination, Jan. 20, 2005 */
#include <stdio.h>
#include <math.h>
// #include <malloc.h>
#define NP 20
#define MP 20
#define MAXSTR 80 // length of string to be read from input file
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;} // inter-change

int* ivector(long n) // make a 1-D flexible dimension array
{ /* allocate n integer*/
    int* pv;
    pv = (int*)malloc((size_t)((n + 1) * sizeof(int)));
    if(!pv)
        printf("allocation failure in ivector()\n");
    return pv;
}

void free_ivector(int* pv)
{ free(pv);
}
Practice 11  Gauss-Jordan Elimination

float** matrix(long u, long v) //make a 2-D flexible dim. array
{
    /* allocate a float matrix with range m[1..u][1..v] */
    long i, nrow=u,ncol=v;
    float** m;
    /* allocate pointers to rows */
    m = (float**)malloc((size_t)((u + 1) * sizeof(float*)));
    if(!m)
        printf("allocation failure 1 in matrix()\n");
    /* allocate rows and set pointers to them */
    m[1] = (float*)malloc((size_t)((u+1) * (v+1))*sizeof(float*));
    if(!m[1])
        printf("allocation failure 2 in matrix()\n");
    for(i=2; i<=u; i++)
        m[i] = m[i-1] + v;
    return m;
}
void free_matrix(float** m)
{
    /* free a float matrix allocated by matrix() */
    free(m[1]);
    free(m);
}
void gaussj(float **a, int n, float **b, int m)
/* a: input matrix, b: input right-hand vector, 
   In output: a: replaced by inverse matrix, b: solutions set */
{
    int *indxc,*indxr,*ipiv;  //3 int arrays for processing pivoting
    int i,icol,irow,j,k,l,ll;
    float big,dum,pivinv,temp;
    indxc =ivector(n);
    indxr =ivector(n);
    ipiv =ivector(n);
    for (j=1;j<=n;j++) ipiv[j]=0;  //main loop for cols to be reduced
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++) {  //loop for searching full pivot element
            if (ipiv[j] != 1)
                for (k=1;k<=n;k++) {
                    if (ipiv[k] == 0) {
                        if (fabs(a[j][k]) >= big) {
                            big=fabs(a[j][k]);
                        }
                    } else if (fabs(a[j][k]) == 0) {
                        ipiv[j]=k;
                    }
                }
        }
irow=j;  
icol=k;
}
}
}

ipiv[icol]++;  
if (irow != icol) {  
    //interchange rows
    for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l]);  
    for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l]);
}

indxr[i]=irow;  //now ready to divide pivot
indxc[i]=icol;
if (a[icol][icol] == 0.0) {
    printf("\ngaussj: Singular Matrix\n");
    exit(0); }
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
for (l=1;l<=n;l++) a[icol][l] = a[icol][l]*pivinv;
for (l=1;l<=m;l++) b[icol][l] = b[icol][l]*pivinv;
Practice 11  Gauss-Jordan Elimination

```c
for (ll=1; ll<=n; ll++)  // reduce rows, except pivot
    if (ll != icol) {
        dum = a[ll][icol];
        a[ll][icol] = 0.0;
        for (l=1; l<=n; l++) a[ll][l] = a[ll][l] - a[icol][l]*dum;
        for (l=1; l<=m; l++) b[ll][l] = b[ll][l] - b[icol][l]*dum;
    }
// end of loop of column reduction

for (l=n; l>=1; l--) {  // swap columns for back correct order
    if (indxr[l] != indxc[l])
        for (k=1; k<=n; k++)
            SWAP(a[k][indxr[l]], a[k][indxc[l]]);
}  // we are done
free_ivector(ipiv);
free_ivector(indxr);
free_ivector(indxc);
```
# Practice 11 Gauss-Jordan Elimination

```c
int main(void)
{
    int j,k,l,m,n;
    float **a,**ai,**u,**b,**x,**t;
    char dummy[MAXSTR];
    FILE *fp;

    a = matrix(NP, NP);
    ai = matrix(NP, NP);
    u = matrix(NP, NP);
    b = matrix(NP, MP);
    x = matrix(NP, MP);
    t = matrix(NP, MP);

    if ((fp = fopen("matrx1.dat", "r")) == NULL)
    {
        printf("Data file matrx1.dat not found\n");
        exit(0);
    }

    while (!feof(fp)) {
        fgets(dummy, MAXSTR, fp);
        fgets(dummy, MAXSTR, fp);
        fscanf(fp, "%d %d ", &n, &m);
        fgets(dummy, MAXSTR, fp);

        scanf(fp, "%d %d ", &n, &m);
        fgets(dummy, MAXSTR, fp);
    }

    // Further code...
}
```

---

**matrx1.dat**

**MATRICES FOR INPUT TO TEST ROUTINES**

Size of matrix (N x N), Number of solutions: 3 1

Matrix A:

9.0 3.0 4.0
4.0 3.0 4.0
1.0 1.0 1.0

Solution vectors:

7.0 8.0 3.0
for (k=1;k<=n;k++)
    for (l=1;l<=n;l++) fscanf(fp,"%f ",&a[k][l]);
fgets(dummy,MAXSTR,fp);
for (l=1;l<=m;l++)
    for (k=1;k<=n;k++) fscanf(fp,"%f ",&b[k][l]);

/* print out matrix a */
printf("Matrix A : 
");
for (k=1;k<=n;k++) {
    for (l=1;l<=n;l++) printf("%12.6f",a[k][l]);
    printf("\n");
}

/* save matrices for later testing of results */
for (l=1;l<=n;l++) {
    for (k=1;k<=n;k++) ai[k][l]=a[k][l];
    for (k=1;k<=m;k++) x[l][k]=b[l][k];
}
Practice 11  Gauss-Jordan Elimination

/* invert matrix */
gaussj(ai,n,x,m);
printf("Inverse of matrix A : 
");
for (k=1;k<=n;k++) {
    for (l=1;l<=n;l++) printf("%12.6f",ai[k][l]);
    printf("\n");
}

/* check inverse */
printf("A multiplies A-inverse:
");
for (k=1;k<=n;k++) {
    for (l=1;l<=n;l++) {u[k][l]=0.0;
        for (j=1;j<=n;j++)
            u[k][l] += (a[k][j]*ai[j][l]);
    }for (l=1;l<=n;l++) printf("%12.6f",u[k][l]);
    printf("\n");
}
/* check solutions */
printf("\nCheck equality of both side of equation:\n\n");
printf("%21s %14s\n","left(original)","right(matrix*sol)");
for (l=1;l<=m;l++) {
    printf("vector %2d: \n",l);
    for (k=1;k<=n;k++) {
        t[k][l]=0.0;
        for (j=1;j<=n;j++)
            t[k][l] += (a[k][j]*x[j][l]);
        printf("%8s %12.6f %12.6f\n"," ",
                b[k][l],t[k][l]);
    }
}
printf("Solution:\n");
for (l=1;l<=m;l++) {
    for (k=1;k<=n;k++) {
        if (m==1) printf("x(%d) = %12.6f\n", k,x[k][l]);
    }
}
```c
else {
    printf("x(%d, %d) = %12.6f \n", k, l, x[k][l]);
}
}

printf("---------------------------------
\n");  
printf("press RETURN for next problem:\n");  
(void) getchar();

fclose(fp);
free_matrix(t);
free_matrix(x);
free_matrix(b);
free_matrix(u);
free_matrix(ai);
free_matrix(a);
return 0;
```

End
Practice 11  Gauss-Jordan Elimination

Result:

<table>
<thead>
<tr>
<th>Matrix A:</th>
<th>9.000000</th>
<th>3.000000</th>
<th>4.000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.000000</td>
<td>3.000000</td>
<td>4.000000</td>
</tr>
<tr>
<td></td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Inverse of matrix A:

<table>
<thead>
<tr>
<th></th>
<th>0.200000</th>
<th>-0.200000</th>
<th>-0.000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.000000</td>
<td>-1.000000</td>
<td>4.000000</td>
</tr>
<tr>
<td></td>
<td>-0.200000</td>
<td>1.200000</td>
<td>-3.000000</td>
</tr>
</tbody>
</table>

A multiplies A-inverse:

<table>
<thead>
<tr>
<th></th>
<th>1.000000</th>
<th>0.000000</th>
<th>0.000001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.000000</td>
<td>1.000000</td>
<td>0.000001</td>
</tr>
<tr>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Check equality of both side of equation:

left (original)  right (matrix*sol)

vector 1:

<table>
<thead>
<tr>
<th></th>
<th>7.000000</th>
<th>7.000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8.000000</td>
<td>8.000000</td>
</tr>
<tr>
<td></td>
<td>3.000000</td>
<td>3.000000</td>
</tr>
</tbody>
</table>

Solution:

x(1) = -0.200000
x(2) =  4.000001
x(3) = -0.800001

press RETURN for next problem:
...

http://gwp01.t.u-tokyo.ac.jp/kouryu/students/classes/ica.htm

**Input file:**

`matx1.dat.txt`

**source of beginning part:**

`Pract11-begin.c.txt`
**LU decomposition**

A procedure for decomposing an $N \times N$ matrix $A$ into a product of a lower triangular matrix $L$ and an upper triangular matrix $U$:

$$A = L U$$

$$
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
= 
\begin{bmatrix}
  l_{11} & 0 & 0 \\
  l_{21} & l_{22} & 0 \\
  l_{31} & l_{32} & l_{33}
\end{bmatrix}
\begin{bmatrix}
  u_{11} & u_{12} & u_{13} \\
  0 & u_{22} & u_{23} \\
  0 & 0 & u_{33}
\end{bmatrix}
$$

$Ax = b$, $L(Ux) = b$, Let $Ux = y$, $\iff$ $Ly = b$$\iff$ $Ux = y \rightarrow x$
5. Linear Algebraic Eqs.
LU decomposition

Demonstration

http://www.cse.uiuc.edu/eot/modules/linear_equations/gaussian_elimination/
Fundamental Numerical Analysis E

(8) Jan. 27, 2005

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Bo Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
## Content of Numerical Algorithm

1. **Introduction:**  
   Error, Accuracy and Stability  
   *(4) 12/9*

2. **Roots Finding**

3. **Interpolation and extrapolation**  
   *(5) 12/16*

4. **Numerical Integration**  
   *(6) 1/13*

5. **Linear algebraic equations**  
   *(7) 1/20*

6. **Nonlinear equation sets**  
   *(8) 1/27*

---

**Now**

**Please login to your Unix ...**
5. Nonlinear Equation Sets

**Linear Equation**

\[ f(x) = 3x - 7 = 0 \]

**Nonlinear Equation**

\[ f(x) = 3x^2 + 4x = 0 \]
\[ 5.0*\sin(x) - x - e^x = 0 \]

**1-Dimension Root Finding**

**Newton-Raphson Method**

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]
5. Nonlinear Equation Sets

Linear Algebraic Equation Sets

A set of linear algebraic equations looks like this:

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N}x_N &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N}x_N &= b_2 \\
    a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3N}x_N &= b_3 \\
    \vdots & \quad \vdots \\
    a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots + a_{MN}x_N &= b_M
\end{align*}
\]

Example of nonlinear equation sets:

\[
\begin{align*}
    x^2 - y - 1 &= 0 \\
    -x + y^2 - 1 &= 0
\end{align*}
\]
5. Nonlinear Equation Sets

Linear Algebraic Equation Sets

Gauss-Jordan Elimination

\[
\begin{bmatrix}
    a_{11} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\
    a_{21} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\
    \vdots & \ddots & \vdots & \ddots & \ddots & \ddots & \ddots \\
    a_{n1} & \cdots & a_{nn} & 0 & 0 & \cdots & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
    1 & 0 & \cdots & 0 & b_{11} & \cdots & b_{1n} \\
    0 & 1 & \cdots & 0 & b_{21} & \cdots & b_{2n} \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
    0 & 0 & \cdots & 1 & b_{n1} & \cdots & b_{nn}
\end{bmatrix}
\]

Elimination:

\[
a_{ij}' = a_{ij} - a_{ik} a_{kj} / a_{kk}
\]

```c
void gaussj(float **a, int n, float **b, int m)
/* a: input matrix, b: input right-hand vector,
   In output: a: replaced by inverse matrix, b: solutions set */
```
5. Nonlinear Equation Sets

Problem Setting

2-Dimension

To solve 2 equations simultaneously:

\[ f(x, y) = 0 \quad \rightarrow \quad (x, y) \]
\[ g(x, y) = 0 \]

N-Dimension

\[ F_i(x_1, x_2, \ldots, x_N) = 0 \quad i = 1, 2, \ldots, N. \quad \rightarrow \quad (x_1, x_2, \ldots, x_N) \]
5. Nonlinear Equation Sets

Example

\[ x^2 - y - 1 = 0 \]
\[ -x + y^2 - 1 = 0 \]

One Root:

\[ x = 1.6180 \]
\[ y = 1.6180 \]
5. Nonlinear Equation Sets

Method for Solving Nonlinear Equation Sets

• **Newton-Raphson**
  - **Simplest** multidimensional root finding,
  - effective if have good initial guess.
  - **Main Drawback:** global convergence

• **Globally convergent method**
  - Backtracking
  - Multidimensional Secant method
  - More advanced implementation
Multidimensional Newton-Raphson Method

\[ F_i(x_1, x_2, \ldots, x_N) = 0 \quad i = 1, 2, \ldots, N. \]  

\( x = (x_1, x_2, \ldots, x_n) \): vector of variables, \( F \): vector of function.

In the neighborhood of \( x \),

\( F \) can be expended in Taylor series:

\[ F_i(x + \delta x) = F_i(x) + \sum_{j=1}^{N} \frac{\partial F_i}{\partial x_j} \delta x_j + O(\delta x^2). \]

\[ J_{ij} \equiv \frac{\partial F_i}{\partial x_j} \quad \rightarrow \quad F_i(x + \delta x) = F_i(x) + \sum_{j=1}^{N} J_{ij} \delta x_j + O(\delta x^2). \]
5. Nonlinear Equation Sets

\[ J_{ij} = \frac{\partial F_i}{\partial x_j} \quad \text{: Jacobian Matrix} \]

\[ y \equiv \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} \quad \begin{cases} y_1 = f_1(x_1, \ldots, x_n) \\ \vdots \\ y_n = f_n(x_1, \ldots, x_n) \end{cases} \]

\[ J(x_1, \ldots, x_n) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{bmatrix} \]
5. Nonlinear Equation Sets

By neglecting \( O(\delta x^2) \),

By setting

\[ F_i(x + \delta x) = 0, \quad F_i(x) + \sum_{j=1}^{N} J_{ij} \delta x_j = 0. \]

\[ \sum_{j=1}^{N} J_{ij} \delta x_j = -F_i(x) \tag{3} \]

Eq. (3) is a set of linear equation sets of correction \( \{ \delta x_j \} \)

With coefficient matrix of \( J_{ij} \). Can be solved by

Gaussian Elimination, LU, … methods.

Eq. (3) \( \rightarrow \) \( \{ \delta x \} \) \( \rightarrow \) \( x_{\text{new}} = x_{\text{old}} + \delta x \)

“Newton step/N-R step”

“Newton step” is a descent direction of \( |F|^2 \)
5. Nonlinear Equation Sets

**Multi-D N-R**

**Procedure of N-R Steps**

\[
\sum_{j=1}^{N} J_{ij} \delta x_j = -F_i(x)
\]

\[
\sum F_i < \Delta_1?
\]

\[
\sum \delta x_i < \Delta_2?
\]

\[
x_{\text{new}} = x_{\text{old}} + \delta x
\]

**Convergent criteria:**
- Sum of abstract of \( F_i < \Delta_1 \)
- Sum of abstract of \( \delta x_i < \Delta_2 \)
- Finish maximum loop
5. Nonlinear Equation Sets  Multi-D N-R

Demonstration of Multidimensional N-R Method

http://www.cse.uiuc.edu/eot/modules/nonlinear_eqns/Newton2D/
Drawbacks of Newton-Raphson Method  
(1-Dimensional as example)

- Divergence at inflection points
- Division of zero or near zero
- Root jumping
- Oscillations
5. Nonlinear Equation Sets

Method for Solving Nonlinear Equation Sets

- **Newton-Raphson:**
  - **Simplest** multidimensional root finding,
  - effective if have good initial guess.
  - **Drawback:** global convergence,
    Jacobian (derivative).

- **Globally convergent method**
  - **Backtracking:** ~ step length
  - Multidimensional Secant method (Broyden’s)
    ~ approximating derivative
  - More advanced implementation
Solve nonlinear equations by Newton-Raphson method:

\[ x + 2y - 2 = 0 \]
\[ x^2 + 4y^2 - 4 = 0 \]

Construction “usrfun”:

Functions:
\[ F_1(x_1, x_2) = x_1 + 2x_2 - 2 \]
\[ F_2(x_1, x_2) = x_1^2 + 4x_2^2 - 4 \]

Jacobian:
\[ J_{11} = \frac{\partial F_1}{\partial x_1} = 1 \quad J_{12} = \frac{\partial F_1}{\partial x_2} = 2 \]
\[ J_{21} = \frac{\partial F_2}{\partial x_1} = 2x_1 \quad J_{22} = \frac{\partial F_2}{\partial x_2} = 8x_2 \]
Practice 12  N-R for nonlinear equations

“Pract12.c”: 4 functions + 1 ancillary function

- Define a user’s function: `usrfun`

- Multidimensional Newton-Raphson method: `mnewt`
  (call “gaussj”)

- perform calculation: `main`

- Gauss-Jordan elimination: `gaussj` (same as last class)

- Ancillary: flexible dimensional array/matrix: “myutil.c”
  (same as last class, put it in same directory as
  “Prac12.c”)

array: `ivector/free_ivector` (integer)

  `vector/free_vector` (float)

matrix: `matrix/free_matrix`
#include <stdio.h>
#include <math.h>  
#include "myutil.c"

void usrfun(float *x, int n, float *fvec, float **fjac) { /* define user's nonlinear equations */
{
   /* Function: */
   /* Jacobian: */
   fjac[1][1] = 1.0;
   fjac[1][2] = 2.0;
   fjac[2][1] = 2.0 * x[1];
   fjac[2][2] = 8.0 * x[2];
}

Practice 12  Multidimensional N-R method
void mnewt(float dx[], float x[], int n)
/* 1 Newton-Raphson step for multidimensional root finding
input: n dimensional vector x;
output: correction dx, updated x */
{
    void gaussj(float**a, int n, float**b, int m);
    int k,i;
    float*fvec,**fjac,**p;
    p=matrix(n,n);
    fvec =vector(n);
    fjac =matrix(n,n);
    usrfun(x,n,fvec,fjac);
    for (i=1;i<=n;i++) p[i][1] = -fvec[i];
    gaussj(fjac,n,p,1);
    for (i=1;i<=n;i++) {
        dx[i]=p[i][1]; //only 1 right-hand vector “b”
        x[i] += p[i][1];
    }
    free_matrix(p); free_vector(fvec); free_matrix(fjac);
}  //end mnewt
#define NTRIAL 10
#define TOLX 1.0e-6
#define N 2
#define TOLF 1.0e-6
#define TINY 1.0e-20
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}

int main(void)
/* To evoke NTRIAL N-R steps of mnewt */
{
    int i,j,k,kk;
    float xx,*dx,*x,*fvec,**fjac,emx,enf;

    fjac = matrix(N,N);
    fvec = vector(N);
    dx = vector(N);
    x = vector(N);
    printf("Input initial vector:\ n");
    for(i=1;i<=N;i++) {
        printf(" x[%d] = ",i);
        scanf("%f",&x[i]);
    }
}
printf("Initial vector is: \n");
    for (i=1;i<=N;i++) {
        printf(" x[%d] = %5.2f \n", i, x[i]);
    }

printf("\n");
for (j=1;j<=NTRIAL;j++) {
    mnewt(dx,x,N);
    usrfun(x,N,fvec,fjac);
    printf("Ntrial: %3d \n", j);
    printf("%5s %13s %13s \n","i","x[i]","f");
    for (i=1;i<=N;i++)
        printf("%5d %16.8f %16.8f \n",i,x[i],fvec[i]);
    // sum of abs. of function:
    errf=0.0;
    for (i=1;i<=N;i++) errf += fabs(fvec[i]);
    if (errf <= TOLF) break;
// sum of correction:

erx=0.0;

    for (i=1;i<=N;i++)  erx += fabs(dx[i]);
    if (erx <= TOLX) break;

    printf("\ npress RETURN to continue...\ n");
    getchar();
    }

    free_vector(dx);
    free_vector(x);
    free_vector(fvec);
    free_matrix(fjac);
    return 0;

} // end main
Practice 12  N-R for nonlinear equations

“Prac12.c”: 4 functions + 1 ancillary function

- Define a user’s function: usrfun
- Multidimensional Newton-Raphson method: mnewt
- perform calculation: main

http://gwp01.t.u-tokyo.ac.jp/kouryu/students/classes/ica.htm
- Gauss-Jordan elimination: gaussj (same as last class)
- Ancillary: flexible dimensional array/matrix: “myutil.c”
  (same as last class, put it in same directory as “Prac 12.c”)
  array: ivector/free_ivector (integer)
  vector/free_vector (float)
  matrix: matrix/free_matrix
void gaussj(float **a, int n, float **b, int m) // same as last time
/* a: input matrix, b: input right-hand vector;
   In output: a: replaced by inverse matrix, b: solutions set. */
{
    int *indxc,*indxr,*ipiv; // 3 int arrays for processing pivoting
    int i,icol,irow,j,k,l,ll;
    float big,dum,pivinv,temp;
    indxc = ivector(n);
    indxr = ivector(n);
    ipiv = ivector(n);
    for (j=1;j<=n;j++) ipiv[j]=0;  // main loop for cols to be reduced
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)  // loop for searching full pivot element
            if (ipiv[j] != 1)
            for (k=1;k<=n;k++) {
                if (ipiv[k] == 0) {
                    if (fabs(a[j][k]) >= big) {
                        if (fabs(a[j][k]) >= big) {
                            big=fabs(a[j][k]);
                        }
                    }
                }
            }
    }
}
irow=j;
icol=k;
}
}
++(ipiv[icol]);
if (irow != icol) { //interchange rows
    for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
    for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
}
indxr[i]=irow; //now ready to divide pivot
indxc[i]=icol;
if (a[icol][icol] == 0.0) {
    printf("\n gaussj: Singular Matrix\ n");
    exit(0);
}
pivinv=1.0/a[icol][icol];
a[icol][icol]=1.0;
for (l=1;l<=n;l++) a[icol][l] *= pivinv;
for (l=1;l<=m;l++) b[icol][l] *= pivinv;
for (ll=1; ll <= n; ll++) // reduce rows, except pivot
    if (ll != icol) {
        dum = a[ll][icol];
        a[ll][icol] = 0.0;
        for (l=1; l <= n; l++) a[ll][l] -= a[icol][l] * dum;
        for (l=1; l <= m; l++) b[ll][l] -= b[icol][l] * dum;
    }
}                        // end of loop of column reduction

for (l=n; l >= 1; l--) {  // swap columns for back correct order
    if (indxr[l] != indxc[l])
        for (k=1; k <= n; k++)
            SWAP(a[k][indxr[l]], a[k][indxc[l]]);
}                     // we are done
free_ivector(ipiv);
free_ivector(indxr);
free_ivector(indxc);
/* put in same directory as “Pract12.c” */
/* Pract12 ancillary, myutil.c  Jan. 20, 2004 */
#include <stdio.h>
#include <math.h>
#include <malloc.h>

int* ivector(long n)        //make a 1-D flexible dimension array
{       /* allocate n integer*/
    int* pv;
    pv = (int*)malloc((size_t)((n + 1) * sizeof(int)));
    if(!pv)
        printf("allocation failure in ivector() \n");
    return pv;
}

void free_ivector(int* pv)
{
    free(pv);
}
float* vector(long n)
{
    /* allocate n floating */
    float* pu;
    pu = (float*)malloc((size_t)((n + 1) * sizeof(float)));
    if(!pu)
        printf("allocation failure in vector()\n");
    return pu;
}

void free_vector(float* pu)
{
    free(pu);
}
float** matrix(long u, long v) // make a 2-D flexible dim. array
{
    /* allocate a float matrix with range m[1..u][1..v] */
    long i, nrow=u,ncol=v;
    float** m;
    /* allocate pointers to rows */
    m = (float**)malloc((size_t)((u + 1) * sizeof(float*)));
    if(!m)
        printf("allocation failure 1 in matrix()\n");
    /* allocate rows and set pointers to them */
    m[1] = (float*)malloc((size_t)((u+1) * (v+1))*sizeof(float));
    if(!m[1])
        printf("allocation failure 2 in matrix()\n");
    for(i=2; i<=u; i++)
        m[i] = m[i-1] + v;
    return m;
}

void free_matrix(float** m)
{
    /* free a float matrix allocated by matrix() */
    free(m[1]);
    free(m);
}
Run:

tying@as301> a.out
Input initial vector:
  x[1] = 1
  x[2] = 2
Initial vector is:
  x[1] = 1.00
  x[2] = 2.00

Ntrial: 1
  i     x[i]     f
  1    -0.8333334   0.0000001
  2     1.4166667   4.7222233

press RETURN to continue...

Ntrial: 2
  i     x[i]     f
  1    -0.1893939   0.0000001
  2     1.0946970   0.8293161

press RETURN to continue...

Ntrial: 3
  i     x[i]     f
  1    -0.0150791   -0.0000001
  2     1.0075395   0.8293161

press RETURN to continue...

Ntrial: 4
  i     x[i]     f
  1    -0.0001120   -0.0000001
  2     1.0000560   0.0004483

press RETURN to continue...

Ntrial: 5
  i     x[i]     f
  1    -0.0000000   -0.0000000
  2     1.0000000   0.0000000
Practice 12: Try more ...

1) \[ x^2 - y - 1 = 0 \]
   \[ -x + y^2 - 1 = 0 \]

2) \[ x^2 + y^2 - 2 = 0 \]
   \[ e^{(x-1)} + y^3 - 2 = 0 \]
Solve nonlinear equations by multidimensional Newton-Raphson method:

\[-x_1^2 - x_2^2 - x_3^2 + x_4^2 = 0\]
\[x_1^2 + x_2^2 + x_3^2 + x_4^2 - 1 = 0\]
\[x_1 - x_2 = 0\]
\[x_2 - x_3 = 0\]
Request on Exercises

- Subject of the email should include message like:
  “FNA-exe1”, “FNA-exe2”, ...
- Subject or Body should include
  “Student Number” if you don’t use ECC account.
- You should Run the program before sending.

Email to both of us:

ying@q.tu-tokyo.ac.jp
zheng@zzz.tu-tokyo.ac.jp
Fundamental Numerical Analysis E

(9) Feb. 3, 2005

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Bo Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
## Content of Numerical Algorithm

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error, Accuracy and Stability</td>
<td></td>
</tr>
<tr>
<td>2. Roots Finding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Interpolation and extrapolation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Numerical Integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Linear algebraic equations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Nonlinear equation sets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Differential Equation / Questionnaire</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now

Please login to your Unix ...
7. Differential Equation

Mathematical Preliminary

A Differential Equation is an equation involving an unknown function and its derivatives.

The Order of the differential equation is the order of the highest derivative of the unknown function involved in the equation.

A Ordinary Differential Equation (ODE) includes a function and its derivatives respect to single variable.

\[ F(x, y, y', y'', \ldots, y^{(n)}) = 0 \]

A Partial Differential Equation (PDE) is an equation involving unknown functions and their partial derivatives.

\[ F(x, y, \ldots, \frac{\partial^2 \psi}{\partial x^2}, \frac{\partial \psi}{\partial z}, \ldots, ) = 0 \]
7. Differential Equation

Examples

- **Newton's Cooling Law:**
  
  \[ \frac{dT}{dt} = -K(T - T_s) \]

  \( T \): temperature of substance at time \( t \)
  
  \( T_s \): surrounding temperature
  
  \( K \): cooling constant

- **Wave Equation:**
  
  \[ \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} + \frac{\partial^2 \psi}{\partial z^2} = \frac{1}{v^2} \frac{\partial^2 \psi}{\partial t^2} \]

  \( \psi \): wave function
  
  \( \psi(x, y, z, t) \)

  \( x, y, z \): position vector variables
  
  \( t \): time variable
Problem Setting

Problem involving ODEs can always be reduced to the sets of 1st-order differential equations.

Eg.

\[
\begin{align*}
\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} &= r(x) \\
\frac{dy}{dx} &= z(x) \\
\frac{dz}{dx} &= r(x) - q(x)z(x)
\end{align*}
\]

Procedure: set \( z \) as an auxiliary variable

\[(2) \rightarrow z' = f(x) \rightarrow z(x)\]
\[(1) \rightarrow y' = z(x) \rightarrow y(x)\]
7. Differential Equation

**Problem Setting**

The generic problem in ODE is thus reduced to the a set of $N$ coupled 1st-order ODE for the functions $y_i$

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, \ldots, y_N), \quad i = 1, \ldots, N$$

**Initial value problem**: all the $y_i$ are given at some starting value $x_s$, and it is desired to find the $y_i$'s at some final point $x_f$.

**Two points boundary value problem**: boundary conditions are specified at more than one $x$. 
7. Differential Equation

Basic: 1st order ODE

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

Initial value: \( y(x_0) = y_0 \)

Geometric interpretation of differential equation: In \((x,y)\) plane, to find a curve which passes point \((x_0, y_0)\) from a group of curves with \(f\) as derivative at \((x,y)\).
7. Differential Equation

Basic Idea of Numerical Algorithm of ODE

Differential equation $\rightarrow$ difference equation

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0$$

**Discrete:**

$$dx \rightarrow \Delta x = x_{i+1} - x_i = h,$$
$$dy \rightarrow \Delta y = y_{i+1} - y_i = F(x, y)$$

**Problem $\rightarrow$ To calculate:** $y_0, y_1, \ldots, y_n$ at $x_0, x_1, \ldots, x_n$.
7. Differential Equation

Numerical Methods

**Euler Method**: conceptually important;
**Runge-Kutta** Method: “workhorse”;

- 4th order
- Adaptive stepsize control for Runge-Kutta

**Richardson extrapolation** (Bulirsch-Stoer);
**Predictor-Corrector** methods;

......
7. Differential Equation

Euler Method

Taylor’s series expansion:

\[ y_{i+1} = y_i + hy'_i + \frac{h^2}{2!} y''_i + \ldots \]

\[ y_{i+1} = y_i + hf(x_i, y_i) + O(h^2) \]

As the 1st order approximation, Euler Formula:

\[ y_{i+1} = y_i + hf(x_i, y_i) \]

Error: \( O(h^2) \)

Euler method assume:
Slope of \( y \) is constant in
The interval \( x_i - x_{i+1} \).
Consider the differential equation

\[ y' = x - y \]

Calculate the solution in x-range: (-1,3) with passing through the point (-1,4).

Comparing to the exact solution is:

\[ y(x) = -1 + 6e^{-(1 + x)} + x \]
/* Practice 13, differential equ, Euler 2005.2.3. */
#include <stdio.h>
#include <math.h>
#define X1 -1.0    // set range X1-X2
#define X2 3.0
#define Y1 4.0    // initial point (X1,Y1)
#define Nstep 21   // number of intervals

float derivs(float x, float y) // set differential equation
{  
    float dydx;
    dydx = x - y;
    return dydx;
}

float Euler(float x, float y, float dydx, float h, float (*diffFunc)(float,float))
{
    return y+h(*diffFunc)(x, y);
}
Practice 13  Euler Method

```c
int main(void)  //calculate y in range of [X1, X2]
{
    float h, x, y, dydx, yout, exact, diff;
    int i;
    y = Y1;
    h = (X2 - X1) / (Nstep-1);
    printf("\n i           x            Euler          Exact        diff\n       n\n\n");
    for(i=0; i<Nstep; i++) {
        x = X1 + i * h;
        dydx = derivs(x, y);
        yout = Euler(x, y, dydx, h, derivs);
        printf("%2d    %12.4f     %12.8f", i, x, y);
        exact = -1.0 + 6.0 * (exp(-1.0 - x)) + x;
        diff = fabs(y-exact);
        printf("       %12.8f  %12.6f\n", exact, diff);
        y = yout;            //advance y
    }
    return 0;
}
```

End
## Practice 13 Euler Method

**Run:**

<table>
<thead>
<tr>
<th>i</th>
<th>x</th>
<th>Euler</th>
<th>Exact</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1.0000</td>
<td>4.00000000</td>
<td>4.00000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>1</td>
<td>-0.8000</td>
<td>3.00000000</td>
<td>3.11238456</td>
<td>0.112385</td>
</tr>
<tr>
<td>2</td>
<td>-0.6000</td>
<td>2.24000001</td>
<td>2.42192030</td>
<td>0.181920</td>
</tr>
<tr>
<td>3</td>
<td>-0.4000</td>
<td>1.67199993</td>
<td>1.89286971</td>
<td>0.220870</td>
</tr>
<tr>
<td>4</td>
<td>-0.2000</td>
<td>1.25759995</td>
<td>1.49597371</td>
<td>0.238374</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.96607995</td>
<td>1.20727670</td>
<td>0.241197</td>
</tr>
<tr>
<td>6</td>
<td>0.2000</td>
<td>0.77286398</td>
<td>1.00716519</td>
<td>0.234301</td>
</tr>
<tr>
<td>7</td>
<td>0.4000</td>
<td>0.65829122</td>
<td>0.87958181</td>
<td>0.221291</td>
</tr>
<tr>
<td>8</td>
<td>0.6000</td>
<td>0.60663295</td>
<td>0.81137908</td>
<td>0.204746</td>
</tr>
<tr>
<td>9</td>
<td>0.8000</td>
<td>0.60530639</td>
<td>0.79179335</td>
<td>0.186487</td>
</tr>
<tr>
<td>10</td>
<td>1.0000</td>
<td>0.64424515</td>
<td>0.81201172</td>
<td>0.167767</td>
</tr>
<tr>
<td>11</td>
<td>1.2000</td>
<td>0.71539611</td>
<td>0.86481899</td>
<td>0.149423</td>
</tr>
<tr>
<td>12</td>
<td>1.4000</td>
<td>0.81231689</td>
<td>0.94430774</td>
<td>0.131991</td>
</tr>
<tr>
<td>13</td>
<td>1.6000</td>
<td>0.92985356</td>
<td>1.04564154</td>
<td>0.115788</td>
</tr>
<tr>
<td>14</td>
<td>1.8000</td>
<td>1.06388283</td>
<td>1.16486037</td>
<td>0.100978</td>
</tr>
<tr>
<td>15</td>
<td>2.0000</td>
<td>1.21110630</td>
<td>1.29872239</td>
<td>0.087616</td>
</tr>
<tr>
<td>16</td>
<td>2.2000</td>
<td>1.36888504</td>
<td>1.44457328</td>
<td>0.075688</td>
</tr>
<tr>
<td>17</td>
<td>2.4000</td>
<td>1.53510809</td>
<td>1.60023975</td>
<td>0.065132</td>
</tr>
<tr>
<td>18</td>
<td>2.6000</td>
<td>1.70808649</td>
<td>1.76394248</td>
<td>0.055856</td>
</tr>
<tr>
<td>19</td>
<td>2.8000</td>
<td>1.88646924</td>
<td>1.93422461</td>
<td>0.047755</td>
</tr>
<tr>
<td>20</td>
<td>3.0000</td>
<td>2.06917548</td>
<td>2.10989380</td>
<td>0.040718</td>
</tr>
</tbody>
</table>

**save as data file:** edata
Practice 13  Euler Method

Accuracy

\%gnuplot
>set xrange [-1:3]
>plot -1.0+6.0*(exp(-1.0-x))+x
>replot "edata" u 2:3

Then calculate
N=60 → edata2
N=100 → edata3
N=200, ...

Euler method needs small step size!
7. Differential Equation  Runge-Kutta Method

Improving Euler formula?

\[ y_{i+1} = y_i + \theta h y_i'(x_i + \theta h) \]

\[ (0 < \theta < 1) \]

\[ y_i'(x_i + \theta h) : \text{average slope in} \ (x_i, x_{i+1}) \]
Runge-Kutta Method

Propagating a solution over an interval by combining the information from several Euler-style steps to match a Taylor series expansion up to certain higher order.

Runge-Kutta formulism

1st order: Euler formula, 1 point
\[ y_{i+1} = y_i + hf(x_i, y_i) + O(h^2) \]

2nd order: Trapezoidal formula, 2 points
\[ y_{i+1} = y_i + \frac{h}{2} [f(x_i, y_i) + f(x_{i+1}, y_{i+1})] + O(h^3) \]

3rd order: 3rd-order Runge-Kutta, 3 points
\[ k_1 = hf(x_i, y_i), \quad k_2 = hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \]
\[ k_3 = hf(x_i + h, y_i - hk_1 + hk_2) \]
\[ y_{i+1} = y_i + \frac{h}{6} (k_1 + 4k_2 + k_3) + O(h^4) \]

Euler: 1
Trapezoidal: 1, 4
3rd-order Runge-Kutta: 1, 2, 4
4th-order Runge-Kutta: 1, 2, 3, 4
4th-order Runge-Kutta: 4 points

\[ k_1 = hf(x_i, y_i) \]
\[ k_2 = hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \]
\[ k_3 = hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2) \]
\[ k_4 = hf(x_i + h, y_i + k_3) \]
\[ y_{i+1} = y_i + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \]
/* Practice 14, differential equ, Runge-Kutta 2005.2.3. */
#include <stdio.h>
#include <math.h>
#define X1 -1.0 // set range X1-X2
#define X2 3.0
#define Y1 4.0 // initial point (X1,Y1)
#define Nstep 21 // number of intervals

float derivs(float x, float y) // set differential equation
{
    float dydx;
    dydx = x - y;
    return dydx;
}

float RungeKutta(float x, float y, float dydx, float h, float (*diffFunc)(float, float));

int main(void) // calculate y in range of [X1, X2]
{ ... // “Euler” in prac13.c -> “RungeKutta” }
**Practice 14  Runge-Kutta method**

```c
float RungeKutta(float x, float y, float dydx, float h, float (*diffFunc)(float,float)) /* Given values for y and dy/dx known at x, use 4th-order Runge-Kutta formula to advance the solution at x+h */
{
    float hh, k1, k2, k3, k4;

    hh = h * 0.5;
    k1 = h * (*diffFunc)(x, y);
    k2 = h * (*diffFunc)(x + hh, y + k1 / 2.0);
    k3 = h * (*diffFunc)(x + hh, y + k2 / 2.0);
    k4 = h * (*diffFunc)(x + h, y + k3);
    return y + (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
}
```

*End*
### Practice 14  Runge-Kutta method

**Run:**

<table>
<thead>
<tr>
<th>i</th>
<th>x</th>
<th>Runge-Kutta</th>
<th>Exact</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1.0000</td>
<td>4.0000000000</td>
<td>4.00000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>1</td>
<td>-0.8000</td>
<td>3.11240005</td>
<td>3.11238456</td>
<td>0.000015</td>
</tr>
<tr>
<td>2</td>
<td>-0.6000</td>
<td>2.42194557</td>
<td>2.42192030</td>
<td>0.000025</td>
</tr>
<tr>
<td>3</td>
<td>-0.4000</td>
<td>1.89280094</td>
<td>1.89286971</td>
<td>0.000031</td>
</tr>
<tr>
<td>4</td>
<td>-0.2000</td>
<td>1.49600780</td>
<td>1.49597371</td>
<td>0.000034</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>1.20731139</td>
<td>1.20727670</td>
<td>0.000035</td>
</tr>
<tr>
<td>6</td>
<td>0.2000</td>
<td>1.00719941</td>
<td>1.00716519</td>
<td>0.000034</td>
</tr>
<tr>
<td>7</td>
<td>0.4000</td>
<td>0.87961441</td>
<td>0.87958181</td>
<td>0.000033</td>
</tr>
<tr>
<td>8</td>
<td>0.6000</td>
<td>0.81140965</td>
<td>0.81137908</td>
<td>0.000031</td>
</tr>
<tr>
<td>9</td>
<td>0.8000</td>
<td>0.79182148</td>
<td>0.79179335</td>
<td>0.000028</td>
</tr>
<tr>
<td>10</td>
<td>1.0000</td>
<td>0.81203735</td>
<td>0.81201172</td>
<td>0.000026</td>
</tr>
<tr>
<td>11</td>
<td>1.2000</td>
<td>0.86484206</td>
<td>0.86481899</td>
<td>0.000023</td>
</tr>
<tr>
<td>12</td>
<td>1.4000</td>
<td>0.94432837</td>
<td>0.94430774</td>
<td>0.000021</td>
</tr>
<tr>
<td>13</td>
<td>1.6000</td>
<td>1.04565978</td>
<td>1.04564154</td>
<td>0.000018</td>
</tr>
<tr>
<td>14</td>
<td>1.8000</td>
<td>1.16487658</td>
<td>1.16486037</td>
<td>0.000016</td>
</tr>
<tr>
<td>15</td>
<td>2.0000</td>
<td>1.29873657</td>
<td>1.29872239</td>
<td>0.000014</td>
</tr>
<tr>
<td>16</td>
<td>2.2000</td>
<td>1.44458556</td>
<td>1.44457328</td>
<td>0.000012</td>
</tr>
<tr>
<td>17</td>
<td>2.4000</td>
<td>1.60025036</td>
<td>1.60023975</td>
<td>0.000011</td>
</tr>
<tr>
<td>18</td>
<td>2.6000</td>
<td>1.76395166</td>
<td>1.76394248</td>
<td>0.000009</td>
</tr>
<tr>
<td>19</td>
<td>2.8000</td>
<td>1.93423271</td>
<td>1.93422461</td>
<td>0.000008</td>
</tr>
<tr>
<td>20</td>
<td>3.0000</td>
<td>2.10990071</td>
<td>2.10989380</td>
<td>0.000007</td>
</tr>
</tbody>
</table>

**Save as data file:** rkdata
Comparing Euler and 4th R-K:

%gnuplot
>set xrange [-1:3]
>plot -1.0+6.0*(exp(-1.0-x))+x
>replot "edata" u 2:3
>replot "rkdata" u 2:3

Runge-Kutta gives good result even in rather large step size!
On Final Assignment

Two questions (50% of final score)
Set out: One today, Another next week

Requests:

1) Run your program and get normal result

2) Send a C code file and a MS Word file (or PDF) as project report: algorithm, program design, test results and analysis.

3) All by email to: ying@q.tu-tokyo.ac.jp
   zheng@zzz.tu-tokyo.ac.jp

4) Deadline: Feb. 24 (also 4 home exercises)
Notice

You will be given no mark if:

• Your program does not work;
• You missed the deadline;
• Your codes are unreasonably similar to others’.
Final Assignment 1

Write a program to solve a 4x4 linear algebraic equations set by both Gaussian Elimination and LU decomposition.

1) The coefficient matrix and right-hand vector should be inputted from an input file;
2) Solve one problem with unique solution set;
3) Considering exception such as $\det | = 0$.
4) Compare these two methods.
Fundamental Numerical Analysis E

(10) Feb. 10, 2005

Lecturer: Ying CHEN
ying@q.t.u-tokyo.ac.jp
Department of Quantum Engineering and System Science

Teaching Assistant: Po Zheng
zheng@zzz.t.u-tokyo.ac.jp
Department of Applied Physics
**Content of Numerical Algorithm**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction: Error, Accuracy and Stability</td>
<td>(4) 12/9</td>
</tr>
<tr>
<td>2. Roots Finding</td>
<td></td>
</tr>
<tr>
<td>3. Interpolation and Extrapolation</td>
<td>(5) 12/16</td>
</tr>
<tr>
<td>4. Numerical Integration</td>
<td>(6) 1/13</td>
</tr>
<tr>
<td>5. Linear Algebraic Equations</td>
<td>(7) 1/20</td>
</tr>
<tr>
<td>6. Nonlinear Equations</td>
<td>(8) 1/27</td>
</tr>
<tr>
<td>7. Differential Equation / Questionnaire</td>
<td>(9) 2/3</td>
</tr>
<tr>
<td>8. Monte Carlo Method</td>
<td>(10) 2/10</td>
</tr>
</tbody>
</table>

**Today**

*Now, Please login to your Unix ...*
8. Monte Carlo Method

- **A statistical methods** by using sequences of **random numbers**.
  
  Also called Random sampling, Statistic simulation, Statistic experiment.

- Apply to **numerical calculation**:
  
  Multiply Integral, Equations' solution, ...

- Apply to **simulation** of Probabilistic processes.
8. Monte Carlo Method

eg. Monte Carlo simulation of physical system
8. Monte Carlo Method

• **Probability distribution function (pdf)** - the physical/mathematical system described by a set of pdf’s.

• **Random number generator** - a source of random numbers with uniform distribution on the unit interval.

• **Sampling rule** - a prescription for sampling from the specified pdf’s.

• **Scoring** - outcomes must accumulate into overall scores for quantities of interest.

• **Error estimation** - measure of the statistical error (variance) as a function of the number of trials.
8. Monte Carlo Method

Probability distribution function (pdf)

Uniform Deviates: 
\[ p(x)dx = \begin{cases} 
  \frac{dx}{y} & 0 < x < 1 \\
  0 & \text{otherwise}
\end{cases} \]

Transformation method

An arbitrary desired probability distribution: \( y(x) \)

| \( p(y)dy \) | = | \( p(x)dx \) | \( \rightarrow \) \( p(y) = p(x)\left|\frac{dx}{dy}\right| \)

Find desired pdf: 
\[ \frac{dx}{dy} = f(x) \quad , \quad x = F(y) \quad \rightarrow \quad y(x) = F^{-1}(x) \]

Inverse of \( F \)
Exponential deviates

\[ p(y)dy = \left| \frac{dx}{dy} \right| dy = e^{-x} \, dy \]

\[ y(x) = -\ln(x) \]

Normal (Gaussian) deviates

\[ p(y)dy = \frac{1}{\sqrt{2\pi}} \, e^{-\frac{y^2}{2}} \, dy \]

\[ y(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]
8. Monte Carlo Method Random Number

Linear Congruential Generator

\[ I_{j+1} = (aI_j + c) \mod m \]

\( I_0, I_1, I_2, \ldots, \) each between 0 and \( m-1; \)

\( a: \) Multiplier \quad \( c: \) Increment \quad \( m: \) modulus

\( m \) can be a very large number, RAND_MAX

e.g. the largest integer of machine:

\[ 2^{15}=32768, \quad \text{or} \quad 2^{31}=2147483648 \]
8. Monte Carlo Method

Random Number

eg. Linear Congruential Generator

\[ a=5 \]
\[ c=1 \]
\[ m=16 \]
\[ x_0=1 \]

\[ I_{j+1} = (5I_j + 1) \mod 16 \]

1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, ...

Initial seed

Random numbers circle in period 16
8. Monte Carlo Method

System-supplied random number

\texttt{rand(\ }): return random integer 0\sim\texttt{RAND\_MAX}

(the same sequence);

\texttt{srand(seed);} … \texttt{rand(\ )}

The \texttt{srand(seed)} function uses the argument seed as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to \texttt{rand()}. 
Practice 15  Random number generator

Create 20 random numbers in [0,1) with 6 places of decimals.

```c
#include<stdio.h>
#include<time.h>
#define RAND_MAX 2147483648 // 2^{31}
main()
{  int i;
    srand(time(NULL));
    for(i=0;i<20;i++)
    {    printf("%8.6f\n",(double)rand()/RAND_MAX);
    }
}
```

use “time( )” as seed.
time(NULL): total seconds passed from 00:00:00 Jan. 1,1970 (UCT)

Run: 0.729026
  0.496963
  0.501846
  0.362712
  ...

8. Monte Carlo Method     M-R Integration (1)

Basic Monte Carlo Strategy

\[ I = \int_{a}^{b} f(x) \, dx \]

To think average of \( f \) over \([a,b] \):

\[ I \approx (b-a) \bar{f} = \frac{b-a}{N} \sum_{i=1}^{N} f(x_i) \]

\( \bar{f} \) is evaluated by \( N \) values at random \( x_i \).

Variance in \( I \):

\[ \sigma_I^2 = \frac{(b-a)^2}{N} \sigma_f^2 = \frac{(b-a)^2}{N} \left[ \frac{1}{N} \sum_{i=1}^{N} f_i^2 - \left( \frac{1}{N} \sum_{i=1}^{N} f_i \right)^2 \right] \]

\( \sigma_I \) small \( \rightarrow \) \( N \) large, \( \sigma_f \) small.
Calculate integration by Monte Carlo Method, Also evaluate the variance.

\[ I = \int_{0}^{1} \frac{1}{1 + x^2} \, dx = \frac{\pi}{4} = 0.785398 \]

\[ N = 10, 100, 1000, 2000, 5000, 10000. \]
/* Pract16, Monte Carlo integration, Feb. 10, 2005 */
#include <stdio.h>
#include <math.h>
#include <time.h>
#define RAND_MAX 2147483648
#define A 0.0
#define B 1.0

/* function for integration*/
double func(double x) {
    return 1.0 / (1.0 + x * x );
}

int main(void) {
    double x, sum, sum2, s, integral, sigma;
    int i, n, m;
    int N[6] = {10, 100, 1000, 2000, 5000, 10000};
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define RAND_MAX 32767

int main()
{
    float A = 0.0;
    float B = 1.0;
    float integral = 0.0;
    float sigma = 0.0;
    float sum = 0.0;
    float sum2 = 0.0;
    int m;
    int n;

    srand(time(NULL));
    printf(" Exact value of integration is 0.785398.\n\n");
    printf(" Points      Integration        Sigma\n");
    for(m=0; m<6; m++)  {
        n = N[m];
        sum = 0;
        sum2 = 0;
        for(i=1; i<=n; i++)  {
            x = A + ((double)rand() / RAND_MAX) * (B - A); //uniform pdf
            s = func(x);
            sum += s;
            sum2 += s * s;
        }
        sum = sum / n; //average of f
        sum2 = sum2 / n;
        integral = sum * (B - A);
        sigma = sqrt((sum2 - (sum * sum)) / n) * (B - A);
        printf("%8d        %10.6f      %10.6f\n", n, integral, sigma);
    }
    return 0;
}
```

End
### Run:

<table>
<thead>
<tr>
<th>Points</th>
<th>Integration</th>
<th>Sigma</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.773294</td>
<td>0.015158</td>
</tr>
<tr>
<td>1000</td>
<td>0.785263</td>
<td>0.004954</td>
</tr>
<tr>
<td>2000</td>
<td>0.784694</td>
<td>0.003621</td>
</tr>
<tr>
<td>5000</td>
<td>0.788294</td>
<td>0.002267</td>
</tr>
<tr>
<td>10000</td>
<td>0.781879</td>
<td>0.001618</td>
</tr>
<tr>
<td>20000</td>
<td>0.786370</td>
<td>0.001133</td>
</tr>
</tbody>
</table>

Exact value: 0.785398

Increasing $N$! Not efficient
Improving Efficiency by Reducing Variance

Considering a positive weight function \( w(x) \):
\[
\int_{a}^{b} w(x) \, dx = 1
\]

\[
I = \int_{a}^{b} f(x) \, dx = \int_{a}^{b} w(x) \frac{f(x)}{w(x)} \, dx
\]

Change variable from \( x \) to \( y(x) \):
\[
y(x) = \int_{0}^{x} w(x') \, dx', \quad dy = w(x) \, dx
\]

Then,
\[
I = \int_{a}^{b} \frac{f(x(y))}{w(x(y))} \, dy \approx \frac{b-a}{N} \sum_{i=1}^{N} \frac{f(x(y_i))}{w(x(y_i))}
\]

Averaging \( f/w \) in \([a,b]\) at uniform pdf in \( y \) → Benefit:

Uniform pdf of points in \( y \) implies distribution of points in \( x \) is \( dy = w(x) \, dx \) → points are more in \( x \) values where \( w \) is large → smaller variance.
eg. Practice 16, Reduction of Variance

\[ f = \frac{1}{1 + x^2} \]

\[ w = \frac{4 - 2x}{3} \]

\[ y = \int w(x)dx = \frac{x(4-x)}{3} \]

\[ x = 2 - \sqrt{4 - 3y} \]

\[ \frac{f}{w} = \frac{1}{1 + x^2} \frac{3}{4 - 2x} \]

Effective!
N=200 now reaches N=2000 accuracy before reduction variance.
To integrate a function over a complicated domain

D: complicated domain.
D': Simple domain, superset of D.

Picking random points over D':
Counting: N: points over D
N': points over D'

\[
\frac{V(\text{or } A)_D}{V(\text{or } A)_{D'}} = \frac{N}{N'}
\]
The probability of a random point lying inside the unit circle:

\[ P \left( x^2 + y^2 < 1 \right) = \frac{A_{\text{circle}}}{A_{\text{square}}} = \frac{\pi}{4} \]

If pick a random point N times and M of those times the point lies inside the unit circle:

\[ P^0 \left( x^2 + y^2 < 1 \right) = \frac{M}{N} \]

if N becomes very large, \( P = P^0 \rightarrow \pi = \frac{4 \cdot M}{N} \)

Exercise 4 (Submit by Feb. 17, 2005)

Create a program to calculate \( \pi \) by Monte Carlo Method (testing various N: large number).
Exercise 4 (Submit by Feb. 17, 2005)

Create a program to calculate $\pi$ now!
(testing various $n$: large number)

**Result:**

Calculate $\pi$ by Monte Carlo Method

- $n=10000$, $\pi=3.1388$
- $n=100000$, $\pi=3.1452$
- $n=1000000$, $\pi=3.14164$
- $n=10000000$, $\pi=3.1422784$
- ...
On Final Assignment

Two questions (50% of final score)
Set out: One today, Another next week

Requests:
1) Run your program and get normal result
2) Send a C code file and a MS Word file (or PDF) as project report: algorithm, program design, test results and analysis.
3) All by email to: ying@q.t.u-tokyo.ac.jp zheng@zzz.t.u-tokyo.ac.jp
4) Deadline: Feb. 24 (also 4 home exercises)
Notice

You will be given **no mark if:**

- Your program does not work;
- You missed the deadline;
- Your codes are unreasonably similar to others'.
Consider a differential equation:
\[ y' = 3x - \sin x \]
\[ y(0) = -1.0 \]

1) \( h=0.5 \), evaluate (without coding):
\[ y(0.5)=? \quad y(1.0)=? \]
by using 4th-order Rung-Kutta formula.

2) Write a program to calculate the solution in \( x \)-range: \((0, \pi)\) by using 4th-order Rung-Kutta formula.

3) Compare to the exact solution, to see \( h=? \) to reach the accuracy of \( 10^{-5} \)?
Closing the Lecture

Summary

- Introduction of Unix
- Review of C programming
- Basic concept, Error, Accuracy and Stability
- 7 Typical topics of numerical analysis

Thanks Mr. Zheng for nice TA!

Thank you very much!
Exercise 3  (Submit by Feb. 9, 2004)

Solve nonlinear equations by multidimensional Newton-Raphson method:

\[-x_1^2 - x_2^2 - x_3^2 + x_4^2 = 0\]
\[x_1^2 + x_2^2 + x_3^2 + x_4^2 - 1 = 0\]
\[x_1 - x_2 = 0\]
\[x_2 - x_3 = 0\]
Request on Exercises

- Subject of the email should include message like:
  "FNA-exe3"
- Subject or Body should include "Student Number" if you don’t use ECC account.
- You should Run the program before sending.

Email to both of us:
  ying@q.t.u-tokyo.ac.jp
  zheng@zzz.t.u-tokyo.ac.jp
Exercise 4  (submit by Jan. 24, 2005)

Based on Practice 14 and Practice 15,
1) Create a GUI to calculate the value of
\[ f(x) = cx^2 + a\sin(bx) \]
by keyboarding \(a\), \(b\), \(c\), and variable \(x\), show result;
2) Display the curve of function in same interface.
3) Re-calculate \(f(x)\) & Redraw the curve of \(f(x)\)
by inputting new parameters (Use “myobject.updateUI();”)
(add “myPanel1.updateUI();” near the end of page 19)
Request on Exercises

- Subject of the email should include "CP-IE-exec4".
- Subject or Body should include "Student Number" if you don’t use ECC account.
- You should Run the program before sending.

Email to us by Jan. 24, 2005

ying@q.t.u-tokyo.ac.jp
yangzl@tkl.iis.u-tokyo.ac.jp
**On Final Assignment**

**Two questions (50% of final score)**
Set out: One today, Another next week

**Requests:**

1) Run your program and get normal result

2) Send a C code file and a MS Word file (or PDF) as project report: algorithm, program design, test results and analysis.

3) All by email to: ying@q.tu-tokyo.ac.jp
   zheng@zzz.tu-tokyo.ac.jp

4) **Deadline: Feb. 24 (also 4 home exercises)**
Notice

You will be given no mark if:

• Your program does not work;
• You missed the deadline;
• Your codes are unreasonably similar to others'.
Final Assignment 1

Write a program to solve a 4x4 linear algebraic equations set by both Gaussian Elimination and LU decomposition.

1) The coefficient matrix and right-hand vector should be inputted from an input file;

2) Solve one problem with unique solution set;

3) Considering exception such as det | A | =0.

4) Compare these two methods.
On Final Assignment

Two questions (50% of final score)
Set out: One today, Another next week

Requests:
1) Run your program and get normal result
2) Send a C code file and a MS Word file (or PDF) as project report: algorithm, program design, test results and analysis.
3) All by email to: ying@q.t.u-tokyo.ac.jp
   zheng@zzz.t.u-tokyo.ac.jp
4) Deadline: Feb. 24 (also 4 home exercises)
Notice

You will be given no mark if:

• Your program does not work;
• You missed the deadline;
• Your codes are unreasonably similar to others’.
Final Assignment 2

Consider a differential equation:
\[ y' = 3x - \sin x \]
\[ y(0) = -1.0 \]

1) \( h=0.5 \), evaluate (without coding):
\[ y(0.5) = ? \quad y(1.0) = ? \]
by using 4th-order Rung-Kutta formula.

2) Write a program to calculate the solution in 
\( x \)-range: \((0, \pi)\) by using 4th-order Rung-Kutta formula.

3) Compare to the exact solution, to see \( h=? \)
to reach the accuracy of \( 10^{-5} \)?
Exercise 1  (Submit by Dec. 22)

Find all roots of following function in [-5,5]:

$$10\sin(x) - x = 0$$

1) Write a program to do pre-search roots;
2) To find all roots in [-5,5] by using Bisection, Secant, False Position, Newton-Raphson method, respectively, at accuracy = 0.000001;
3) Take one root as example, comparing the convergent speed of 4 method. Summarize the results on to a WORD file.
Request on Exercises

- Subject of the email should include message like:
  “FNA-exe1”, “FNA-exe2”, ...
- Subject or Body should include
  “Student Number” if you don’t use ECC account.
- You should Run the program before sending.

Email to both of us:

ying@q.tu-tokyo.ac.jp
zheng@zzz.tu-tokyo.ac.jp
Exercise 2  (Submit by Jan. 12, 2005)

There is a 10 points data set:

<table>
<thead>
<tr>
<th>x:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>y:</td>
<td>1</td>
<td>8</td>
<td>27</td>
<td>64</td>
<td>125</td>
<td>216</td>
<td>343</td>
<td>512</td>
<td>729</td>
<td>1000</td>
</tr>
</tbody>
</table>

Calculate the \(y\)-value at \(x=2.5\), \(5.7\) and \(9.75\) by using cubic spline interpolation method.

Marry Christmas and Happy New Year!

See you on Jan. 13, 2005!
Request on Exercises

- Subject of the email should include message like:
  
  “FNA-exe1”, “FNA-exe2”, ...

- Subject or Body should include
  
  “Student Number” if you don’t use ECC account.

- You should Run the program before sending.

Email to both of us:

ying@q.tu-tokyo.ac.jp
zheng@zzz.tu-tokyo.ac.jp