EDV2
Wissenschaftliches Programmieren

Inhalt

- Einführung in das Programmieren mit FORTRAN77
  (2 Einheiten)
    - Elementare Befehle
    - Funktionen

Erste Übung

- Arbeitsgruppen (max. 2 Personen / Gruppe)

- Login: `edv2mi01, edv2mi02`, ...

- Passwort: steht an der Tafel
  Eine Gruppe nach der anderen ändern mit: `passwd`
  neues Passwort merken!!

- Übungsverzeichnis: `01Ue2012-03-06`
  ist bereits erstellt.

Unterlagen

`/home/EDV2/Unterlagen/`

FORTRAN

- Es gibt viele Programmiersprachen
  Auswahl hängt von vielen Kriterien ab.

- FORTRAN:
  seit vielen Jahrzehnten Programmiersprache in der Physik

  -> es existieren umfangreiche Programmbibliotheken

- Compiler auf Rechengeschwindigkeit optimiert

- Viele Programme existieren in FORTRAN
  Sie müssen verstanden, adaptiert und erweitert werden

Hinweise zum Programmieren

- Zuerst das Problem verstehen,
  dann die Programmstruktur überlegen, dann programmieren
- So einfach als möglich
- Ausführlich kommentieren und dokumentieren
- Schleifen einrücken
- Leerzeilen einfügen

Formatierung des Quelltextes

Fortran wurde in den 1960 Jahren für die Eingabe mittels Lochkarten entwickelt

- Die Spalten 7-72 enthalten die Anweisungen
- Spalte 1: C (oder !) Kennzeichnet eine Kommentarzeile
- Spalten 2-5: Anweisungsnummern (Labels)
- Spalte 6: markiert Fortsetzungszeile
- (73–80: Lochkartennummern)

Variablentypen in FORTRAN

`INTEGER*2`	2-Byte Integer
`INTEGER = INTEGER*4`	4-Byte Integer
`INTEGER*8`	8-Byte Integer
`REAL = REAL*4`	4-Byte Real
`DOUBLE PRECISION = REAL*8`	8-Byte Real
`COMPLEX = COMPLEX*8`	2x4-Byte Real
`DOUBLE COMPLEX = COMPLEX*16`	2x8-Byte Real
`LOGICAL`	`.TRUE., .FALSE.`
`CHARACTER`	1-Byte Zeichen `'A' = "A"`
`CHARACTER*20`	20-Byte Zeichenkette `'Ax' = "Ax"`

Konversionen

```
INT(X), REAL(I), DBLE(I), CMPLX(X, Y)
```

Bedingte Anweisungen

```
IF (X .EQ. 0.) Y = 1.
```

```
IF (X .EQ. 0.) THEN
Y = 1.
Z = -1.
END IF
```

```
IF (X .EQ. 0.) THEN
Y = 0.
ELSE IF (X .LT. 0.) THEN
Y = -1.
ELSE
Y = 1.
END IF
```

Vergleichsoperatoren

`==`	`.EQ.`	gleich "equal"
`!=`	`.NE.`	ungleich "not equal"
`<`	`.LT.`	kleiner "less than"
`<=`	`.LE.`	kleiner gleich "less than or equal"
`>`	`.GT.`	größer "greater than"
`>=`	`.GE.`	größer gleich "greater than or equal

Schleifen

```
      DO 10 I=1,N,2
        ....
 10   CONTINUE
```

```
      DO I=N,1,-2
      ....
      END DO
```

```
      DO WHILE (I .LE. N)
      ....
      END DO
```

```
      WRITE (*,'(5F10.6)') (X(I), I=1,10,2)
```

Felder

C:

```
int i[10];      // i[0]...i[9]
double x[100]; // x[0]... x[99]
```

FORTRAN:

```
      INTEGER I(10)
C     I(1)...I(10)

      DOUBLE PRECISION X(100)
C     X(1)...X(100)

      DOUBLE PRECISION X(-2:2)
C     X(-2), X(-1), X(0), X(1),X(2)

      DOUBLE PRECISION X(-2:2,100)
C     X(-2, 1)...X(2, 100)
```

Ein/Ausgabe am Bildschirm/Terminal

```
      PRINT *, 'Das ist x = ', X

      WRITE(*, *) 'Das ist wieder x = ', X

      WRITE(6, *) 'Unit 6 ist der Bildschirm'

      WRITE(*, 100) X, Y
      WRITE(6, 100) X, Y
100   FORMAT(1X, 'X und Y: ', 2F12.6)

      READ(*, *) X, Y
      READ(5, *) X, Y
C Unit 5 ist das Terminal

      READ(*, 110) X, Y
      READ(5, 110) X, Y
110   FORMAT(2F12.6)
```

Potenzierungsoperator

x^y

`pow(x, y)`	in C
`x**y`	in FORTRAN

Funktionen

Verwendung im Hauptprogramm:

```
      PROGRAM NECK
      DOUBLE PRECISION FNECK
      FLAECHE = FNECK(N, R, PI)
      STOP
      END
```

Implementation der Funktion:

```
      DOUBLE PRECISION FUNCTION FNECK(N, R, PI)
      DOUBLE PRECISION N, R, PI
C     Rueckgabewert := Name der Funktion !!
      FNECK = N*R*R*SIN(PI/N)
      RETURN
      END
```

Unterprogramme (Subroutines)

Verwendung im Hauptprogramm:

```
      PROGRAM NECK
      CALL WRNECK(PI)
      STOP
      END
```

Implementation des Unterprogramms:

```
      SUBROUTINE WRNECK(PI)
C     Parameter werden uebergeben
C     Gibt keinen Wert zurueck
      DOUBLE PRECISION PI, F, FNECK
      F = FNECK(12, 1.D0, PI)
      PRINT *, F
      RETURN
      END
```

Felder und Unterprogramme

```
      PARAMETER (N = 4)
      DOUBLE PRECISION X, Y
      DIMENSION X(N), Y(10)
      DATA X /1.d0, -4.d0, 7.d0, 2.d0/
      CALL TEST(X, N, Y, 10)



      SUBROUTINE TEST(F1, N1, F2, N2)
C     F1 ... feld1
C     N1 ... Groesse feld1
C     F2 ... feld2
C     N2 ... Groesse feld2
      DIMENSION F1(N1)
      DIMENSION F2(N2)
       ...
      RETURN
      END
```

Ein-/Ausgabe auf Dateien

```
      OPEN(UNIT=1, FILE='test.dat', ERR=200)
      WRITE(1, *) X, ' ', Y
      WRITE(1, 100) X, Y
      CLOSE(1)
100   FORMAT(1X, G15.6, 3X, G15.6)
      STOP
200   PRINT *, "Fehler beim Oeffnen!"

      READ(1, *) X, Y
      DO
            READ(1, FMT=100, END=300) X, Y
      END DO
300   CONTINUE
```

```
OPEN(UNIT=1, FILE='test.dat')
CALL AUSGABE(X, N)
CALL EINLESEN(X, N)
END

SUBROUTINE AUSGABE(X, N)
WRITE(1, *) (X(I), I=1,N)
RETURN
END


SUBROUTINE EINLESEN(X, N)
READ(1, *) (X(I), I=1,N)
RETURN
END
```

```
INTEGER U
U=1
OPEN(UNIT=1, FILE='test.dat')
CALL AUSGABE(U, X, N)
CALL EINLESEN(U, X, N)
CLOSE(UNIT=1)
END


SUBROUTINE AUSGABE(U, X, N)
WRITE(U, *) (X(I), I=1,N)
RETURN
END


SUBROUTINE EINLESEN(U, X, N)
READ(U, *) (X(I), I=1,N)
RETURN
END
```

3.2.1 Ober- und Untersumme:

Näherungslösung: $f(x) \geq 0$ für alle $x \in [a, b]$

- $[a, b]$ wird in n Teilintervalle zerlegt $x_k,\ k = 1, \ldots n$ und $x_{k-1} < \xi_k < x_k$.
- In jedem Teilintervall $f(x) = C, C = \min(\mathrm{f}(\xi_k)) = f_u(x_k)$ im Teilintervall $k \rightarrow$
- **Untersumme** $U(n, f) = \sum_{k=1}^{n} f_u(x_k) \cdot \Delta x_k$ mit $\Delta x_k = (x_k - x_{k-1})$
- In jedem Teilintervall $f(x) = C, C = \max(\mathrm{f}(\xi_k)) = f_o(x_k)$ im Teilintervall $k \rightarrow$
- **Obersumme** $O(n, f) = \sum_{k=1}^{n} f_o(x_k) \cdot \Delta x_k$

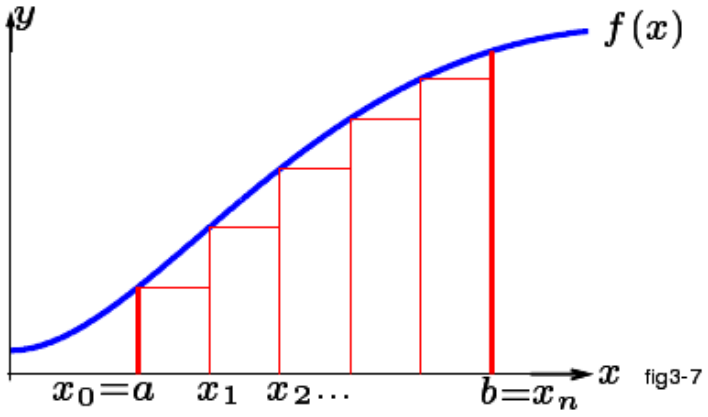


fig3-7

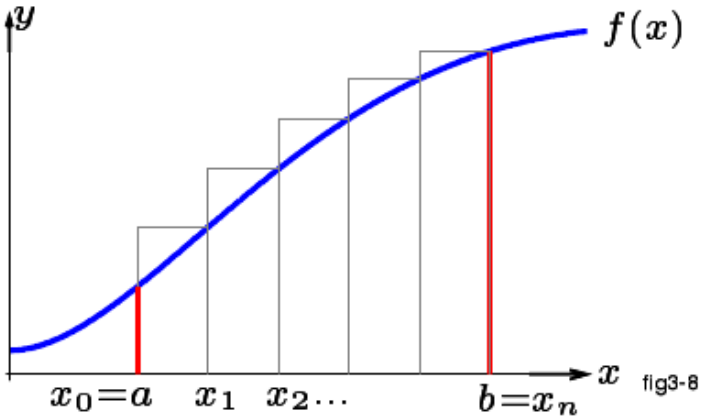


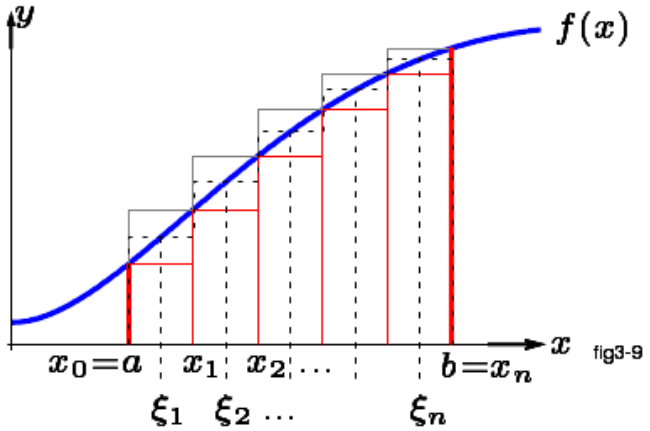
fig3-8

3.2.2 Das Riemannsches Integral:

$$U(n,f) \leq F \leq O(n,f)$$

In jedem Teilintervall wird ein Zwischenpunkt ξ_k gewählt.

n Zwischenpunkte $\xi_1, \xi_2, \ldots, \xi_n$ mit $\xi_k \in [x_{k-1},\ x_k]$



Verfeinerung der Zerlegung: $n \to \infty$

Die maximale Länge aller Teilintervalle $\Delta x_k \to 0$

$\Rightarrow \quad U(\infty,f) = O(\infty,f) = F$ (exakter Wert)

$$F = \lim_{n\to\infty} \sum_{k=1}^{n} f(\xi_k)\cdot(x_k - x_{k-1}) = \lim_{n\to\infty} \sum_{k=1}^{n} f(\xi_k)\cdot\Delta x_k = \int_a^b f(x)\,dx$$

F heißt **Riemannsche Integral** von $f(x)$ über $[a,b]$, wenn der Grenzwert unhabängig von der Zerlegung des Intervalls ist. (Bsp 3.1)

nimmt wieder einen Satz von äquidistanten Gitterpunkten, $x_0(=a),\ldots,x_n(=b)$ mit $x_i = x_0 + ih$ an, die durch eine konstante Schrittweite h voneinander getrennt sind. Die Funktionswerte an den Gitterpunkten (Stützstellen) werden mit $f_i = f(x_i)$ bezeichnet. Je nachdem, ob man die Funktion $f(x)$ zwischen den Stützstellen durch Polynome ersten, zweiten oder dritten Grades ersetzt, nimmt man verschiedene Teilintervalle an und erhält folgende Summenausdrücke

$$\int_a^b dx\ f(x) = \frac{h}{2}\left(f_0 + 2f_1 + 2f_2 + \cdots 2f_{n-1} + f_n\right) - \frac{b-a}{12}h^2 f''(\xi), \tag{5.54}$$

$$\int_a^b dx\ f(x) = \frac{h}{3}\left(f_0 + 4f_1 + 2f_2 + 4f_3 \cdots 2f_{n-2} + 4f_{n-1} + f_n\right) - \frac{b-a}{180}h^4 f^{IV}(\xi), \tag{5.55}$$

$$\int_a^b dx\ f(x) = \frac{3h}{8}\left(f_0 + 3f_1 + 3f_2 + 2f_3 + 3f_4 + 3f_5 \cdots 3f_{n-1} + f_n\right) - \frac{b-a}{80}h^4 f^{IV}(\xi). \tag{5.56}$$

Die jeweils letzten Terme in den obigen Gleichungen erlauben eine Fehlerabschätzung; es gilt jeweils $\xi \in [a,b]$.

Die erste Integrationsregel, Gleichung (5.54), entspricht der summierten Trapezregel; die Zahl der Stützstellen, $n+1$, ist beliebig. Bei der zweiten Regel, der Simpson-Regel, muss n durch zwei teilbar sein, bei der dritten Regel, der Simpson 3/8 Regel, muss n durch drei teilbar sein.