

Numerische Bibliotheken unter UNIX

$$\sum a_i b_i = ?$$

$$\sum c_i d_i = ?$$

$$\sum e_i f_i = ?$$

Numerische Bibliotheken unter UNIX

Frank Elsner
Frank.Elsner@rz.uni-osnabrueck.de
Universität Osnabrück
- Rechenzentrum -
Albrechtstraße 28
D-49076 Osnabrück

Version: 1.1
Stand: 97/01/22
WWW <http://www.rz.uni-osnabrueck.de/rz/doc/scripts/00index.htm>

Inhaltsverzeichnis

1. Einleitung	1
1.1 Abhängigkeiten.....	1
1.2 Verwendete typografische Konventionen	2
1.3 Weiterführende Literatur	2
2. Überblick über numerische Bibliotheken	3
2.1 Einordnen der Bibliotheken.....	3
2.2 Computeralgebrasysteme.....	3
3. libm.a - Standard Mathematik Bibliothek für C und Fortran	4
3.1 Gliedern des Funktionsumfangs	4
3.2 Einbinden in Fortran.....	4
3.3 Einbinden in C.....	5
3.4 Behandeln von Fehlern in Fortran	5
3.5 Behandeln von Fehlern in C	7
4. BLAS - IBM Basic Linear Subroutine Library	8
4.1 Gliedern des Funktionsumfangs	8
4.2 Einbinden in Fortran.....	8
4.3 Einbinden in C.....	9
5. ESSL - IBM Engineering and Scientific Subroutine Library	10
5.1 Gliedern des Funktionsumfangs	10
5.2 Einbinden in Fortran.....	10
5.3 Einbinden in C.....	11
5.4 Behandeln von Fehlern.....	11
6. NAGF - NAG Fortran Library	12
6.1 Gliedern des Funktionsumfangs	12
6.2 Einbinden in Fortran.....	13
6.3 Einbinden in C.....	13
6.4 Behandeln von Fehlern in Fortran	14
6.5 Behandeln von Fehlern in C	14
6.6 Abrufen von Online Informationen.....	14
7. Überblick über Graphik-Bibliotheken und -Programme	15
7.1 Einordnen von Graphik-Bibliotheken und -Programmen	15
7.2 Auflisten der Ausgabegeräte.....	15
8. Überblick über gemischtsprachige Programmierung	17
8.1 Gliedern in Problemfelder	17
8.2 Zuordnen der Datentypen	17
8.3 Übergeben von Argumenten.....	17
8.4 Bearbeiten von Feldern.....	18
8.5 Bearbeiten von Zeichenketten	18
9. Überblick über statische und dynamische Felder	20

9.1 Aufzählen der Sprachkonstrukte.....	20
9.2 Abfragen von Systemparametern.....	21
10. Anhang	22
10.1 math.h.....	22
10.2 esl.h.....	23
10.3 nagf.h.....	23
10.4 make Beschreibungsdatei	23
11. Literaturhinweise	25

1. Einleitung

Dieses Handbuch wendet sich an Programmierer mit dem Schwerpunkt numerische Anwendungen, die auf einem UNIX System mit den Programmiersprachen C und/oder Fortran arbeiten und kommerzielle numerische Bibliotheken in ihren Programmen einbinden wollen.

In diesem Handbuch werden schwerpunktmäßig folgende Fragen behandelt:

- **Welche numerischen Bibliotheken unter UNIX stehen zur Verfügung und wie werden Funktionen aus diesen Bibliotheken in eigene Programme eingebunden?**

Das Handbuch ist folgendermaßen gegliedert:

- Die ersten 5 Kapitel geben einen Überblick über einige wichtige numerische Bibliotheken und Programme. Ausgewählte numerische Bibliotheken werden jeweils in einem eigenen Kapitel vorgestellt.

Die Einbindung der Prozeduren in ein Fortran und/oder C Programm wird dabei exemplarisch durch ein vollständiges Programm und die zugehörige **make** Beschreibungsdatei (*Makefile*) illustriert.

- Das folgende Kapitel gibt einen kurzen Überblick über einige wichtige graphische Bibliotheken und Programme.
- Die folgenden 2 Kapitel befassen sich mit der gemischtsprachigen Programmierung und dynamischen und statischen Feldern in C und Fortran.
- Im Anhang sind Auszüge aus wichtigen C Deklarationsdateien, eine vollständige **make** Beschreibungsdatei und Shell Skripte zur Zeitmessung enthalten.

1.1 Abhängigkeiten

Als Grundlage der Beschreibung dient eine Workstation **IBM RISC/6000** des Rechenzentrums (`titan.rz.Uni-Osnabrueck.DE`) unter dem Betriebssystem **AIX**, einem UNIX Derivat.

Die Beschreibung läßt sich, mit einigen Einschränkungen (z.B. Existenz und Pfadnamen von Kommandos und Bibliotheken, unterschiedliche Compiler und Compiler-Optionen), auch auf andere UNIX Systeme übertragen.

1.2 Verwendete typografische Konventionen

Fettschrift	Fettschrift bezeichnet Definitionen, Kommandos oder wichtige Textpassagen. Beispiel: Das Kommando x1c ruft den C Compiler auf.
<i>Kursivschrift</i>	Kursivschrift bezeichnet englische Fachausdrücke oder Variablen, die durch konkrete Werte zu ersetzen sind. Beispiel: Geben Sie eine beliebige Zahl <i>n</i> ein.
[...]	Eckige Klammern bezeichnen optionale Syntaxelemente. Beispiel: vi [<i>filename</i> ...]
<...>	Spitze Klammern bezeichnen Umschalttasten, die zusammen mit einer weiteren Taste (oder zwei weiteren Tasten) eingegeben werden. Beispiel: Geben Sie die Tastenkombination <CTRL>+V ein.
Courier	Schreibmaschinenschrift bezeichnet Dateinamen oder Programmbeispiele. Dateien sind zusätzlich mit einem Rahmen umgeben. Beispiel: Die mathematische Standardbibliothek heißt /usr/lib/libm.a.
\$... # ...	Das Zeichen \$ symbolisiert die Eingabeaufforderung des Kommandointerpreters, hinter dem Kommandos einzugeben sind. Die Eingabe ist durch Fettschrift ausgezeichnet. Sie müssen Kommandos immer durch <CR> (<i>Enter</i>) bestätigen (ausführen). Hinter dem Zeichen # folgt ein Kommentar mit einer kurzen Erklärung. Beispiel: \$ vi sample5.c # starts the vi editor

1.3 Weiterführende Literatur

In diesem Handbuch werden einige Problemfelder nur angeschnitten, zum anderen sind die Beschreibungen von Kommandos und Bibliotheken aus Platzmangel unvollständig. Abhängig von Ihren konkreten Aufgaben benötigen Sie weiterführende Literatur, z.B. die genaue C und/oder Fortran Sprachbeschreibung oder die Referenzdokumentation für die Bibliotheksfunktionen.

Im Text erfolgt ein Hinweis auf weiterführende Literatur in der Form [*Cn*], wobei sich die Numerierung auf das Kapitel **Literaturhinweise** bezieht. Die genannten Handbücher können in der RZ Bibliothek eingesehen und ggf. kurzfristig entliehen werden.

2. Überblick über numerische Bibliotheken

In diesem Kapitel werden einige wichtige numerische Bibliotheken vorgestellt [A9-16, C1-2, E1-4].

2.1 Einordnen der Bibliotheken

Viele numerische Probleme lassen sich auf elementare numerische Verfahren zurückführen. Falls nicht besondere Gründe für die Entwicklung eigener Prozeduren sprechen, sollten Prozeduren aus bereits vorhandenen und ausgetesteten kommerziellen Bibliotheken verwendet werden.

Folgende numerischen Bibliotheken stehen u.a. zur Verfügung¹:

- **libm.a** - Standard Mathematik Bibliothek für C und Fortran
- **libblas.a** - IBM Basic Linear Subroutine Library
- **libessl.a** - IBM Engineering and Scientific Subroutine Library
- **libnagf.a** - NAG Fortran Library
- **libcplex.a** - CPLEX Optimization Subroutine Library

Die folgende Tabelle listet die Produktbezeichnungen der Bibliotheken, ihre Dateinamen und, falls vorhanden, die zugehörigen C Deklarationsdateien (*C header files*) auf:

Bibliothek	Datei	Zugehörige Deklarationen
Standard Mathematik Bibliothek für C und Fortran	/usr/lib/libm.a	/usr/include/math.h /usr/include/errno.h
IBM Basic Linear Subroutine Library	/usr/lib/libblas.a	-
IBM Engineering and Scientific Subroutine Library	/usr/lib/libessl.a	/usr/include/essl.h
NAG Fortran Library	/usr/local/lib/libnagf.a	/usr/local/include/ nagf.h

In den folgenden Kapiteln wird auf eine vollständige Auflistung der Funktionen verzichtet, vielmehr werden die übergeordneten Anwendungsgebiete aufgezählt.

2.2 Computeralgebrasysteme

Ferner stehen folgende Computeralgebrasysteme zur Verfügung, mit denen sich auch numerische Probleme elegant bearbeiten lassen:

- **Maple**
- **Mathematica**

Interessant sind hierbei insbesondere folgende Möglichkeiten im Bereich der Numerik:

- Durchführen von numerischen Berechnungen mit beliebiger Genauigkeit
- Erzeugen von optimierten C und Fortran Code
- Einfaches Zugreifen auf umfangreiche Bibliotheken math. Funktionen und Verfahren

¹ Die Aufzählung gilt speziell für das RS/6000 Cluster des Rechenzentrums. Erkundigen Sie sich ggf. bei Ihrem Systemverwalter, welche numerischen Bibliotheken installiert sind.

3. libm.a - Standard Mathematik Bibliothek für C und Fortran

In diesem Kapitel wird die Standard Mathematik Bibliothek behandelt, die zum Lieferumfang jedes C und Fortran Compiler gehört [A9-13].

3.1 Gliedern des Funktionsumfangs

Die Standard Mathematik Bibliothek enthält eine Sammlung häufig benötigter mathematischer Funktionen in folgenden Bereichen (siehe auch **Anhang**, `math.h`):

- trigonometrische Funktionen und deren Umkehrfunktionen
- Exponential- und logarithmische Funktionen
- Konvertierungs- und Rundungsfunktionen
- mathematische Konstanten

3.2 Einbinden in Fortran

Das folgende Beispielprogramm `sample1` berechnet den Sinus von ArcusSinus von 0,5:

```
program main
  print *, "sin(asin(0.5)): ", sin(asin(0.5))
end
```

Die folgende **make** Beschreibungsdatei (*Makefile*) enthält alle notwendigen Informationen, um die Fortran Quelle `sample1.f` zu übersetzen. Die externen Referenzen für `sin` und `asin` werden von **xlF** aufgelöst, weil die Bibliothek `libm.a` per Voreinstellung durchsucht wird.

```
F77=x1
FFLAGS=-O

.f:
    $(F77) $(FFLAGS) -o $@ $<

all: sample1
```

Übersetzen Sie nun das Programm und führen Sie es aus:

```
$ make
```

```
$ sample1
```

```
sin(asin(0.5)): 0.5000000000
```

Für alle folgenden Programme werden ebenfalls *Makefile* Dateien verwendet. In den folgenden Fortran **make** Beschreibungsdateien (*Makefiles*) werden nur die Erweiterungen und Änderungen gegenüber der obigen *Makefile* Datei beschrieben.

3.3 Einbinden in C

Die Aufgabe, den Sinus von ArcusSinus von 0,5 zu berechnen, wird nun in einem C Programm sample2 gelöst:

```
#include <stdio.h>

#include <math.h>

int main(void)
{
    printf("sin(asin(0.5)): %f\n", sin(asin(0.5)))
    exit(0);
}
```

```
CC=xlc
CFLAGS=-qlanglvl=ansi -O
LDFLAGS=-lm -lxlf

.c:
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<

all: sample2
```

\$ **sample2**

```
sin(asin(0.5)): 0.5000
```

3.4 Behandeln von Fehlern in Fortran

Das folgende Beispielprogramm sample3 demonstriert, wie fehlerhafte reelle Arithmetik (*Floating Point Exception*, FPE) entdeckt werden kann.

```
block data
include '/usr/include/fpdc.h'
include '/usr/include/fpdt.h'
end

program main
include '/usr/include/fpdc.h'
include '/usr/include/fexcp.h'

! We define variables which represent certain values.
real zero /0.0d0/, one /1.0d0/, huge /1.0d300/, tiny /1.0d-300/

fpstat(fpze) = .true.           ! Division by zero
fpstat(fpoe) = .true.           ! Overflow
fpstat(fpue) = .true.           ! Underflow
fpstat(fpxe) = .true.           ! Invalid
fpstat(fpve) = .true.           ! Inexact

call fpsets(fpstat)
call signal(sigtrap,xl__trce)
! end of FPE detection specific code

! print *, "Dividing by zero: ", one/zero
! print *, "Producing overflow: ", huge * huge
```

```
! print *, "Producing underflow: ", tiny * tiny
  print *, "Dividing zero by zero: ", zero/zero

end
```

```
FFLAGS=-qflttrap -g

all: sample3
```

Das Programm `sample3` reagiert nun auf eine FPE, wie z.B. eine Division der Form $0/0$:

```
$ sample3
```

```
Trap encountered, Traceback:

Offset      12c in procedure main
Offset       0 in procedure start

--- End of call chain ---
```

Eine Analyse mit `dbx` zeigt, welche Programmzeile verantwortlich ist:

```
$ dbx sample3
```

```
(dbx) run
```

```
trace trap in main at line 39
39      print *, "Dividing zero by zero: ", zero/zero
```

```
(dbx) quit
```

Falls Sie das Programm ohne (!) die Option `-qflttrap` übersetzen, wird die FPE ignoriert:

```
$ sample3
```

```
Dividing zero by zero: 0.0000000000E+00
```

Sie können das Melden von Ausnahmesituationen auch folgendermaßen aktivieren:

```
FFLAGS=-qflttrap=\
      overflow:underflow:zerodivide:invalid:inexact:enable -g\
      -qsigtrap

all: sample3
```

3.5 Behandeln von Fehlern in C

In C Programmen wird die globale Variable `errno` bei Auftreten eines Fehlers auf einen Wert gesetzt, dessen Bedeutung in der Datei `<errno.h>` definiert wird. Beachten Sie, daß `errno` sofort nach Beendigung des Funktionsaufrufs ausgewertet werden muß.

Das folgende Beispielprogramm `sample4` demonstriert eine Auswertung von `errno`:

```
#include <stdio.h>
#include <math.h>
#include <errno.h>
int errno;                                /* global error variable */

void main(void)
{
    errno=0;
    printf("log(0): %f\n", log(0));        /* invalid arg. to log */
    switch(errno)
    {
        case EDOM:      printf("%d: Out of Domain.\n", errno);
                        break;
        case ERANGE:    printf("%d: Out of Range.\n", errno);
                        break;
        default:        printf("%d: No error text.\n", errno);
                        break;
    }
    exit(0);
}
```

\$ **sample4**

log(0): -INF

33: Out of Domain.

4. BLAS - IBM Basic Linear Subroutine Library

In diesem Kapitel wird die BLAS Bibliothek (*Basic Linear Algebra Subprograms*) behandelt [A5].

4.1 Gliedern des Funktionsumfangs

Die BLAS Bibliothek enthält Prozeduren zur Durchführung elementarer Operationen der Linearen Algebra. Die BLAS Bibliothek bildet mittlerweile einen internationalen Standard. Wie andere Hersteller bietet auch IBM für Rechnerarchitekturen speziell angepaßte BLAS Bibliotheken. Die BLAS Bibliothek enthält Prozeduren aus folgenden Bereichen:

- Vektor-Vektor-Verarbeitung (Level 1)
- Matrix-Vektor-Verarbeitung (Level 2)
- Matrix-Matrix-Verarbeitung (Level 3)

Die BLAS Bibliothek dient als Grundlage für viele Standardbibliotheken wie NAG Fortran oder ESSL; d.h. Prozeduren aus diesen Bibliotheken rufen Prozeduren aus der BLAS Bibliothek auf. Da zur Sicherheit in den Standardbibliotheken "langsamere" BLAS Prozeduren vorhanden sind, ist die Suchreihenfolge der Bibliotheken beim Binden entscheidend. Die herstellereigene BLAS Bibliothek muß zuerst durchsucht werden und deshalb an erster Stelle stehen (siehe jeweils *Makefile*).

Der Namensgebung der Prozeduren liegt folgende Systematik zugrunde:

- Jeder Operation ist ein Basisname (*root*) zugeordnet (z.B. **DOT** für das Skalarprodukt). Diesem Namen wird ein Buchstabe (*prefix*) vorangestellt (z.B. **S** für *Single Precision*), um den Datentyp zu kennzeichnen. Bei Bedarf wird dem Basisnamen ein Buchstabe (*suffix*) hinzugefügt, um verschiedene Versionen zu unterscheiden.
- Beispielsweise heißt die Prozedur zum Berechnen des Skalarprodukts von Vektoren vom Typ *Single Precision* **SDOT**.

4.2 Einbinden in Fortran

Das folgende Beispielprogramm `sample5` berechnet das Skalarprodukt zweier Vektoren unter Zuhilfenahme der Prozedur `ddot`:

```
program main
double precision r1(3), r2(3)
data r1 /1,1,1/, r2 /1,0,1/
print *, "ddot(3,r1,1,r2,1)", ddot(3,r1,1,r2,1)
end
```

```
LDFLAGS=-lblas
```

```
all: sample5
```

```
$ sample5
```

```
ddot(3,r1,1,r2,1): 1.000000000
```

4.3 Einbinden in C

Das folgende Beispielprogramm `sample6` löst das Problem in C. Da keine vorgefertigen Prototyp-Definitionen für C existieren (vielleicht demnächst: `blas.h`?), müssen diese "in Handarbeit" erstellt werden:

```
#include <stdio.h>

/* prototype for ddot */
extern double ddot(int * pi1, double ar1[], int * pi2,
                   double ar[], int * pi3);

int main(void)
{
    int i1=3, i2=1, i3=1;
    double r1[3]={1,0,1} , r2[3]={1,1,0};
    printf("ddot(3,r1,3,r2,1): %f\n",
           ddot(&i1,r1,&i2,r2,&i3) );
    exit(0);
}
```

```
LDFLAGS=-lm -lxlf -lblas

all: sample6
```

\$ **sample6**

```
ddot(3,r1,1,r2,1): 1.000000000
```

Für die BLAS Bibliothek ist keine eigene Fehlerbehandlung vorgesehen. Es können allerdings die im vorherigen Kapitel genannten Verfahren verwendet werden.

5. ESSL - IBM Engineering and Scientific Subroutine Library

In diesem Kapitel wird die ESSL Bibliothek (*Engineering and Scientific Subroutine Library*) behandelt [A14].

5.1 Gliedern des Funktionsumfangs

Die ESSL Bibliothek enthält eine Sammlung hochoptimierter mathematischer Prozeduren und zusätzlicher Hilfsfunktionen. Sie enthält ca. 140 Prozeduren aus folgenden Bereichen:

- Lineare Algebra
- Matrixoperationen
- Lineare Gleichungssysteme
- Eigenwerte
- Fourier Transformation
- Sortieren und Suchen
- Interpolation
- Numerische Integration
- Zufallszahlengenerierung

Der Namensgebung der Prozeduren liegt folgende Systematik zugrunde:

- ♦ Der erste Buchstabe bezeichnet den erwarteten Datentyp (S, D, C, Z, I für *real*, *double*, *complex*, *double complex* and *integer*). Die folgenden Buchstaben liefern eine Abkürzung für die Aufgabe der Prozedur.
- ♦ Beispielsweise vertauscht **CSWAP** zwei komplexe Vektoren (**C** für *complex*, **SWAP** für Vertauschen).

5.2 Einbinden in Fortran

Das folgende Beispielprogramm `sample7` führt eine numerische Integration über eine Menge von Punkten unter Zuhilfenahme der Prozedur `dptnq` (*Numerical Quadrature Performed on a Set of Points*) durch:

```

program main
integer N, i
parameter(N=5)
real*8 X(N), Y(N), ErrorEstimate, NumInt, ExactInt
data X /1.0, 2.0, 2.5, 3.0, 5.0/
do i=1, N      Y(i)=exp(X(i))
end do
call dptnq(X, Y, N, NumInt, ErrorEstimate)
ExactInt=exp(5.0)-exp(1.0)
print *, "Num. Integration: ", NumInt
print *, "Exact:           ", ExactInt
print *, "Error Estimate:   ", ErrorEstimate
print *, "Error:            ", abs(NumInt-ExactInt)
end

```

```

LDLFLAGS=-lblas -lessl

all: sample8

```

```
$ sample8
  Num. Integration: 151.366633087724779
  Exact:           145.694877274117545
  Error Estimate:  -3.25458841408978694
  Error:           5.67175581360723413
```

5.3 Einbinden in C

Das folgende Beispielprogramm sample9 löst das Problem in C:

```
#include <stdio.h>
#include <math.h>
#include <essl.h>
#define N 5

int main(void)
{
    int      i;
    double   Y[N], ErrorEstimate, NumInt, ExactInt;
    double   X[N] = {1.0, 2.0, 2.5, 3.0, 5.0};

    for (i=0; i < N; i++) { Y[i]=exp(X[i]); };
    dptnq( X, Y, n, &NumInt, &ErrorEstimate); /* rc is void! */
    ExactInt=exp(5.0)-exp(1.0);
    printf("Num. Integration:  %g\n", NumInt);
    printf("Exact:           %g\n", ExactInt);
    printf("Error Estimate:    %g\n", ErrorEstimate);
    printf("Error:           %g\n", abs(NumInt-ExactInt) );
    exit(0);
}

LDLFLAGS=-lm -lxlfe -lblas -lessl

all:  sample9
```

```
$ sample9

  Num. Integration: 151.367
  Exact:           145.695
  Error Estimate:  -3.25459
  Error:           5.67175
```

5.4 Behandeln von Fehlern

Bei Auftreten eines Fehlers in einer ESSL Prozedur wird das Programm mit einer Fehlermeldung abgebrochen. Die Fehlermeldung bezeichnet die genaue Fehlerursache, die möglichen Fehler sind für jede Prozedur dokumentiert. Es besteht die Möglichkeit, ausgewählte Fehler im Programm abzufangen und zu korrigieren.

6. NAGF - NAG Fortran Library

In diesem Kapitel wird die NAG Fortran Bibliothek (*Numerical Algorithm Group*) behandelt [C1-2].

6.1 Gliedern des Funktionsumfangs

Die NAG Fortran Bibliothek ist neben der IMSL Fortran Bibliothek die bekannteste Standardbibliothek für Numerik und Statistik. Sie enthält ca. 1000 Prozeduren, die sich auf folgende Bereiche verteilen:

- A02 - Komplexe Arithmetik
- C02 - Nullstellen von Polynomen
- C05 - Lösungen nichtlinearer Gleichungssysteme
- C06 - Summation und FFT
- D01 - Numerische Integration
- D02 - Gewöhnliche Differentialgleichungen
- D03 - Partielle Differentialgleichungen
- D04 - Numerisches Differenzieren
- D05 - Integralgleichungen
- E01 - Interpolation
- E02 - Kurven- und Flächenanpassung
- E04 - Minimierung und Maximierung von Funktionen
- F01 - Matrixoperationen, Invertierung
- F02 - Eigenwerte und -vektoren
- F03 - Determinanten
- F04 - Simultane Lösung linearer Gleichungssysteme
- F05 - Orthogonalisierung
- F06 - Hilfsfunktionen (Teilmenge von LAPACK)
- F07 - Lineare Gleichungssysteme
- G01 - Einfache Statistiken
- G02 - Korrelation and Regression
- G03 - Multivariate Methoden
- G04 - Varianzanalyse
- G05 - Zufallszahlen
- G07 - Univariate Schätzungen
- G08 - Nichtparametrische Statistik
- G11 - Kontingenztafeln
- G12 - Überlebenssurvival analysis
- G13 - Zeitreihenanalyse
- H - Operations Research
- M01 - Sortierverfahren
- P01 - Fehlermeldungen
- S - Spezielle Funktionen
- X01 - Mathematische Konstanten
- X02 - Maschinenkonstanten
- X03 - Innere Produkte
- X04 - Ein-/Ausgabeprozeduren
- X05 - Datum- und Zeitprozeduren

Der Namensgebung der Prozeduren liegt folgende Systematik zugrunde:

- ♦ Jeder Prozedurname besteht aus 6 Buchstaben. Die ersten 3 bezeichnen das Kapitel (siehe oben); die nächsten 2 die laufende Nummer im Kapitel. Der letzte Buchstabe bezeichnet die Genauigkeit, wobei **F** für *Double Precision* und **E** für *Single Precision* steht.
- ♦ Beispielsweise ist **F02AAF** eine Prozedur aus dem Kapitel **F02**, die Variablen vom Datentyp *Double Precision* (**F**) bearbeitet.

6.2 Einbinden in Fortran

Das folgende Beispielprogramm `sample10` soll die Eigenwerte einer symmetrischen 4x4-Matrix berechnen:

```

program main
  integer nMax
  parameter (nMax=4, nOut=6)
  integer i, iFail /0/
  double precision rA(nMax, nMax), rB(nMax), rC(nMax)
  external f02aaf
  data rA /   0.5,   0.0,   2.3,  -2.6,
&           0.0,   0.5,  -1.4,  -0.7,
&           2.3,  -1.4,   0.5,   0.0,
&          -2.6,  -0.7,   0.0,   0.5 /
  call f02aaf(rA,nMax,nMax,rB,rC,iFail)
  write(nOut,9999) (rB(i),i=1,nMax)
  stop
9999 format (1x,4f9.4)
end

```

```
LDLFLAGS=-L/usr/local/lib -lblas -lessl -lnagf
```

```
all: sample10
```

```
$ sample10
```

```
-3.0000 -1.0000  2.0000  4.0000
```

6.3 Einbinden in C

Das folgende Beispielprogramm `sample11` löst das Eigenwert-Problem in C:

```

#include <stdio.h>
#include <nagf.h>

#define nMAX=4
int main(void)
{
  int iFail=0, i;
  static double ar1[nMAX][nMAX]=
  {
    0.5,  0.0,  2.3, -2.6,
    0.0,  0.5, -1.4, -0.7,
    2.3, -1.4,  0.5,  0.0,
    -2.6, -0.7,  0.0,  0.5 };
  double ar2[nMAX], ar3[nMAX];
  f02aaf(ar1, &nMAX, &nMAX, ar2, ar3, &iFail);
  for (i=1; i <= nMAX; i++)
  {
    printf("%g\t", i, ar2[i-1]);
  }
  printf("\n");
  exit(0);
}

```

```
CFLAGS=-I/usr/local/include
LDFLAGS=-L/usr/local/lib -lm -lxlf -lblas -lessl -lnagf

all: sample11
```

6.4 Behandeln von Fehlern in Fortran

Die Prozeduren der NAGF Bibliothek melden Fehler grundsätzlich über die Statusvariable `IFAIL` an die aufrufende Prozedur. Ein Wert von Null zeigt an, daß kein Fehler aufgetreten ist, Werte ungleich Null zeigen Fehler an, die spezifisch für jede Prozedur dokumentiert sind.

Die Reaktion einer Prozedur auf einen Fehler ist abhängig vom Wert von `IFAIL` vor Aufruf der Prozedur:

- ♦ Falls `IFAIL=0` gesetzt wird, wird im Fehlerfall eine Meldung ausgegeben und das Programm abgebrochen (*hard failure*).
- ♦ Falls `IFAIL=1` gesetzt wird, wird das Programm im Fehlerfall ohne jede Warnung fortgesetzt (*soft failure, no message*).
- ♦ Falls `IFAIL=-1` gesetzt wird, wird das Programm im Fehlerfall nach Ausgabe einer Warnung fortgesetzt (*soft failure*).

6.5 Behandeln von Fehlern in C

Die Fehlerbehandlung in C entspricht der in Fortran; d.h. im C Programm muß die Variable `IFAIL` vor dem Aufruf auf 1, 0 oder -1 gesetzt und nach dem Aufruf ausgewertet werden.

6.6 Abrufen von Online Informationen

Es steht ein interaktives Hilfe-System zur Verfügung:

```
$ setup naghel
$ naghel
```

Sie finden im Verzeichnis `/usr/misc/nag/nagf*/examples` zu fast allen Prozeduren ein vollständiges Beispielprogramm.

7. Überblick über Graphik-Bibliotheken und -Programme

In diesem Kapitel wird ein kurzer Überblick über Graphik-Bibliotheken und -Programme gegeben, mit denen numerische Ergebnisse visualisiert werden können. Ferner werden die möglichen Ausgabegeräte aufgelistet [A20, C3-4, E2-4].

7.1 Einordnen von Graphik-Bibliotheken und -Programmen

Folgende Graphik-Bibliotheken und -Programme stehen u.a. zur Verfügung:

Produkt	Kurzbeschreibung	Aufruf
GNU gnuplot	setup gnuplot man gnuplot	setup gnuplot gnuplot
NAG Graphics Library	setup nag_help naghelp	setup nag_graphics xlf -L/usr/local/lib -lnagg ...
Maple (enthält integrierte Graphik)	setup maple man maple	setup maple [x]maple
Mathematica (enthält integrierte Graphik)	setup mathematica man math	setup mathematica math

7.2 Auflisten der Ausgabegeräte

Sie können Graphiken mit dem **lpr** Kommando auf einem Drucker bzw. Plotter ausgeben:

```
$ lpr -P queue filename
```

Zum Beispiel wird eine PostScript Datei `sample.ps` mit folgendem Kommando auf dem HP Laserjet III SI (PostScript) des RZ gedruckt:

```
$ lpr -P ps2rz sample.ps
```

Folgende Drucker und Plotter stehen u.a. zur Verfügung:

Name (queue)	Typbezeichnung	Format
lprz	IBM 6262	ASCII
pcl2rz	HP Laserjet III Si	PCL 5
pcl1rz	HP DeskJet 500 C	PCL
pl1rz	IBM 6186 DIN A0 Plotter	HPGL
ps1rz	Apple Laserwriter II NTX	PostScript
ps2rz	HP Laserjet III Si PostScript	PostScript
ps2rzdp	HP Laserjet III Si PostScript, doppelseitiger Druck	PostScript

Überblick über Graphik-Bibliotheken und -Programme

Eine Ausnahme bildet der elektrostatische Plotter VERSATEC CE3425E. Eine Plotdatei *fn* im PPM Format (siehe z.B. **man xv**) wird mit dem Kommando **vsplot** in das Versatec Format konvertiert und im Anschluß mit **lpr** geplottet :

```
$ vsplot fn | lpr -Ppunrz
```

Ausgaben können im AVZ, Untergeschoß, Raum B20, abgeholt werden.

8. Überblick über gemischtsprachige Programmierung

In diesem Kapitel werden Problemfelder behandelt, die bei der gemischtsprachigen Programmierung in C und Fortran auftreten. Gemichstsprachige Programmierung ist dann notwendig, wenn große Teile des Programmes in einer Sprache (häufig in C) geschrieben werden, während andere Teile aus vorgefertigten numerischen Bibliotheken (häufig in Fortran) eingefügt werden.

8.1 Gliedern in Problemfelder

Bei gemichstsprachiger Programmierung in C und Fortran, d.h. bei der Verwendung von C Funktionen in Fortran Programmen und von Fortran Funktionen (und Unterprogrammen) in C Programmen, sind folgende Problemfelder zu beachten:

- Korrespondierende Datentypen
- Numerierung und Speicherung von Feldern
- *call by reference* versus *call by value*
- Zeichenketten

8.2 Zuordnen der Datentypen

Die folgende Tabelle beschreibt die Zuordnung zwischen den C und Fortran Datentypen für die Compiler **xlf** und **xlc**:

Datentyp	C	Fortran	Bemerkungen
Zeichen	char	CHARACTER	
Zeichenkette	char[n+1]	CHARACTER*n	
Wahrheitswert	unsigned char	LOGICAL*1	
	short unsigned int	LOGICAL*2	
Ganze Zahl	signed char	INTEGER*1	
	short signed int	INTEGER*2	
	signed int	INTEGER*4	
Reelle Zahl	float	REAL*4 / REAL	
	double	REAL*8 / DP	xlf : REAL*16
Komplexe Zahl	struct CPMLX { float r,i;};	COMPLEX	
	struct DCPLX { double r,i;};	COMPLEX*16	xlf : COMPLEX*32
Struktur	struct xyz {};	-	xlf : Erweiterung
Aufzählung	enum xyz {};	-	
	union xyz {};	EQUIVALENCE	
Zeiger auf Funktion	pointer to function	function argument	
Feld	array[n][m]...	array(n, m,...)	

8.3 Übergeben von Argumenten

In C werden beim Aufruf einer Funktion Argumente per Voreinstellung mit ihrem Wert übergeben (*call by value*), in Fortran mit ihrer Adresse (*call by reference*).

Bei einem Aufruf einer Fortran Funktion in einem C Programm müssen Sie deshalb alle Variablen als Referenzen (Zeiger, *pointer*) übergeben. Insbesondere können Sie Konstanten nicht direkt als Argumente übergeben, sondern müssen vorher den Wert der Konstanten an eine Variable übergeben.

Bei einem Aufruf einer C Funktion in einem Fortran Programm werden alle Argumente als Zeiger übergeben. In der gerufenen C Funktion müssen deshalb alle Argumente als Zeiger deklariert sein.

8.4 Bearbeiten von Feldern

In C werden Felder in **Zeilen-Spalten-Ordnung** im Speicher abgelegt, in Fortran in **Spalten-Zeilen-Ordnung**. C beginnt die Numerierung von Feldelemente und Zeichenketten immer bei 0, Fortran beginnt per Voreinstellung bei 1.

Zum Beispiel entspricht einer reellwertigen 2D-Matrix A_{34} mit $n=3$ Zeilen, Laufindex i , und $m=4$ Spalten, Laufindex j ,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

in C die Definition:

```
#define N 3
#define M 4
float A[N][M];
```

und folgende Reihenfolge im Speicher:

```
A[0][0] -A[0][1] -A[0][2] - ... -
A[1][0] - ...
(Zeile für Zeile, j variiert zuerst)
```

und in Fortran die Definition:

```
parameter (N=3, M=4)
real A(N,M)
```

und folgende Reihenfolge im Speicher:

```
A(1,1) - A(2,1) - A(3,1) - A(1,2) -
...
(Spalte für Spalte, i variiert zuerst)
```

8.5 Bearbeiten von Zeichenketten

In C werden Zeichenketten mit dem Zeichen '`\0`' beendet und können die Länge Null besitzen, da das abschließende Zeichen nicht mitgezählt wird.

Sie müssen deshalb an eine Fortran Zeichenkette zusätzlich ein abschließendes Zeichen '`\0`' anhängen, bevor Sie die Zeichenkette an eine C Funktion übergeben. Sie müssen ggfs. auch den in Fortran reservierten Speicher um 1 erhöhen, damit das zusätzliche Zeichen aufgenommen werden kann.

In Fortran besitzen Zeichenketten mindestens die Länge 1 und in der Regel keine abschließende Null. Bei der Übergabe von Zeichenketten verwendet Fortran intern verborgene letzte Argumente, in denen die Längen 'heimlich', also für den Programmierer nicht sichtbar, übergeben werden.

Beim Aufruf einer Fortran Funktion aus C, die Zeichenketten als Argumente erwartet, müssen deshalb für den `xlc` Compiler in der rufenden C Funktion die Längen der Zeichenketten als zusätzliche letzte Argumente übergeben werden². Zur Berechnung der Länge kann zweckmäßig die C Funktion `strlen()` verwendet werden, hier am Beispielprogramm `sample12` demonstriert:

```
void f77_print(char* chName, int iN); /* 2 arguments */

#include <stdio.h>

int main(void)
{
    char chName[]="ABC";
```

² Die Vorgehensweise ist leider von C Compiler zu C Compiler unterschiedlich!

Überblick über gemischtsprachige Programmierung

```
    f77_print(chName, strlen(chName)); /* extra argument */
    exit(0);
}
```

```
subroutine f77_print(chName)                ! one argument

character*(*) chName                       ! variable length
print *, "Length: ", LEN(chName)          ! LEN = strlen
print *, "chName: ", chName
return
end
```

\$ **sample12**

```
Length: 3
ABC
```

9. Überblick über statische und dynamische Felder

In diesem Kapitel werden Möglichkeiten beschrieben, statische und dynamische Felder in C, C++, Fortran 77 und Fortran 90 zu erzeugen. Die Beschreibung bezieht sich auf die Compiler **xlc**, **xlc**, **xlF** und **f90** [A3, A6, A9-13, A17-18, C6].

9.1 Aufzählen der Sprachkonstrukte

Statische Felder werden bereits zur Übersetzungszeit in ihrer Größe festgelegt und können während der Programmausführung nicht verändert und freigegeben werden. **Dynamische Felder** können zu beliebigen Zeiten und in (fast) beliebiger Größe während der Programmausführung erzeugt und freigegeben werden. Dynamische Felder sind deshalb flexibler zu handhaben als statische Felder und erfordern weniger Systemressourcen.

C, C++ und Fortran 90 unterstützen dynamische Felder, Fortran 77 (ohne Gnade der späten Geburt) nicht.

In den folgenden Beispielprogrammen wird jeweils ein Feld mit 100 Elementen erzeugt, das letzte Element ausgegeben und, nur für dynamische Felder, das Feld wieder freigegeben

Statisches Feld in C und C++:

```
#include <stdio.h>
#define N 100
static double ar1[N];

int main(void)
{
    printf("ar1[N-1]: %g\n", ar1[N-1]);
    exit(0);
}
```

Dynamisches Feld in C und C++:

```
#include <stdio.h>
#define N 100

int main(void)
{
    double *ar1;
    ar1=(double *) calloc(N,sizeof(double));
    printf("ar1[N-1]: %g\n", ar1[N-1]);
    free(ar1);          /* releases memory to operating system.
*/
    exit(0);
}
```

Statisches Feld in Fortran 77 und Fortran 90:

```
program main
parameter (N=100)
double precision ar1(N)
print *, "ar1(N): ", ar1(N)
end
```

Dynamisches Feld in Fortran 90:

```
program main
parameter (N=100)
real (kind=2), dimension(:), allocatable :: ar1
allocate(ar1(N))
print *, "ar1(N): ", ar1(N)
deallocate(ar1)
end
```

9.2 Abfragen von Systemparametern

RISC/6000 Systeme verwenden eine virtuelle Speicherverwaltung, so daß der virtuelle Speicher prinzipiell nur durch die Größe des Auslagerungsbereiches (*swap space*) eingeschränkt ist. Aus Gründen der gleichmäßigen Systemauslastung existieren darüberhinaus benutzerspezifische Einschränkungen für die Nutzung virtuellen Speichers, die durch den Systemverwalter festgelegt werden.

Das Kommando **vmstat** liefert u.a. die aktuelle Verteilung von belegtem und freiem virtuellen Speicher (Einheit: 1 page = 4094 Bytes = 4 KB):

```
$ vmstat
```

```
procs      memory
...        avm  fre      # avm: memory in use
           6421 5210    # fre: free memory
```

Das Kommando **ulimit** zeigt u.a. die benutzerspezifischen Maximalgrößen für virtuellen Speicher und Dateigröße an. Diese Werte können bei nachgewiesenem Bedarf vom Systemadministrator vergrößert werden.

```
$ ulimit -a
```

```
time(seconds) unlimited      # execution time limit
file(blocks)      097151      # file size limit
data(kbytes)      11072      # program data size limit
```

Im Beispiel liegt die maximale Dateigröße bei 209751 Blöcken (ca. 100 MB) und die maximale Größe des Datenbereiches bei 131072 KBytes (128 MB). Ein Programm kann u.a. nicht übersetzt bzw. ausgeführt werden, wenn es als Datei zu groß ist (siehe `file` Parameter) oder wenn es einen zu großen Datenbereich benötigt (siehe `data` Parameter).

Da virtueller Speicher eine der wesentlichen Ressourcen eines Systems darstellt, sollte er von allen Benutzern verantwortungsbewußt genutzt werden. Zum Beispiel sollte dynamischer Speicher sobald als möglich wieder freigegeben werden.

10. Anhang

Dieser Anhang enthält zur Verdeutlichung der jeweiligen Struktur Auszüge aus folgenden C Deklarationsdateien:

- /usr/include/math.h
- /usr/include/essl.h
- /usr/local/include/nagf.h

Ferner beinhaltet er Listings von Skriptdateien zur Zeitmessung und einer vollständigen **make** Beschreibungsdatei.

10.1 math.h

```
extern double acos(double x);
extern double asin(double x);
extern double atan(double x);
extern double atan2(double x, double y);
extern double ceil(double x);
extern double cos(double x);
extern double cosh(double x);
extern double exp(double x);
extern double fabs(double x);
extern double floor(double x);
extern double fmod(double x, double y);
extern double frexp(double value, int *exp);
extern double ldexp(double x, int exp);
extern double log(double x);
extern double log10(double x);
extern double modf(double value, double *iptr);
extern double pow(double x, double y);
extern double sin(double x);
extern double sinh(double x);
extern double sqrt(double x);
extern double tan(double x);
extern double tanh(double x);

#define M_E 2.7182818284590452354E0
#define M_LOG2E 1.4426950408889633870E0
#define M_LOG10E 4.3429448190325181667E-1
#define M_LN2 6.9314718055994530942E-1
#define M_LN10 2.3025850929940456840E0
#define M_PI 3.1415926535897931160E0
#define M_PI_2 1.5707963267948965580E0
#define M_PI_4 7.8539816339744827900E-1
#define M_1_PI 3.1830988618379067154E-1
#define M_2_PI 6.3661977236758134308E-1
#define M_2_SQRTPI 1.1283791670955125739E0
#define M_SQRT2 1.4142135623730951455E0
#define M_SQRT1_2 7.0710678118654752440E-1

...
```

10.2 essl.h

```

/* Definition of complex data types */

typedef union { struct { float __re, __im; }
__data; double __align; } cplx;
typedef union { struct { double __re, __im; }
__data; double __align; } dcplx;

#define RE(x) ((x).__data.__re)
#define IM(x) ((x).__data.__im)

/* Linear Algebra Subprograms */

/* Vector-Scalar Subprograms */

int isamax(int*, float *, int*);
int idamax(int*, double *, int*);
int icamax(int*, cplx *, int*);
int izamax(int*, dcplx *, int*);

...

void srot(int*, float *, int*, float *, int*, float*, float*);
void drot(int*, double *, int*, double *, int*, double*, double*);
void crot(int*, cplx *, int*, cplx *, int*, float*, cplx*);
void zrot(int*, dcplx *, int*, dcplx *, int*, double*, dcplx*);
void csrot(int*, cplx *, int*, cplx *, int*, float*, float*);
void zdrot(int*, dcplx *, int*, dcplx *, int*, double*, double*);

...

```

10.3 nagf.h

```

extern void a00aaf_(
#ifdef __STDC__
void
#endif
);

extern void a02aaf_(
#ifdef __STDC__
double *xxr,
double *xxi,
double *yr,
double *yi
#endif
);

extern double a02abf_(
#ifdef __STDC__
double *xxr,
double *xxi
#endif
);

...

```

10.4 make Beschreibungsdatei

```

#-----#
# Makefile template for C and Fortran programming
# Calling sequence: make -f makefile or simply make
#-----#

# Linker flags

```

Anhang

```
LDFFLAGS=-L/usr/local/lib -lm -lxf -lblas -lessl -lnagf

# Hints for LDFFLAGS:
# Order from bottom to top, put low level libs in front of high level libs.
# Delete names of libs that you don't need.

#-----#
# Fortran:
#-----#

# Fortran Compiler
F77=xf

# Full debugging
FDEBUG=-C -D -qcheck -qextchk -qflttrap -g -qnomaf -qfips -qstat -w -v -u
# Full optimization
FOPTIMIZE=-O3 -Q -Pv -Wp,-ew

# Compiler Flags, either FDEBUG or FOPTIMIZE
FFLAGS=$(FDEBUG)

# Suffix rules
.f.o:
$(F77) $(FFLAGS) -c $<
.f:
$(F77) -o $@ $(FFLAGS) $(LDFFLAGS) $<

#-----#
# C:
#-----#

# C Compiler
CC=xc

# Full debugging and profiling
CDEBUG=-qlanglvl=ansi -g

# Full optimization
COPTIMIZE=-qlanglvl=ansi -O

# Compiler Flags, either CDEBUG or COPTIMIZE
CFLAGS=$(CDEBUG)

# Suffix rules
.c.o:
$(CC) $(CFLAGS) -c $<
.c:
$(CC) -o $@ $(CFLAGS) $(LDFFLAGS) $<

#-----#
# These are the project specific rules. Substitute appropriate values
# for PGM, OBS and LIBS, e.g. PGM=raytrace, OBS=getstring.o view.o ...
# LIBS=libproj1.a.
#-----#

PGM=sample1
OBS=func1.o func2.o
LIBS= proj1

$(PGM): $(PGM).o $(LIBS) $OBS)
$(CC) -o $(PGM) $(LDFFLAGS) $(PGM).o $(OBS) $(LIBS)
```

11. Literaturhinweise

Im Text erfolgt ein Hinweis zum Beispiel mit [B5]. Er bezieht sich auf den Titel "Einführung in UNIX" im Abschnitt B mit der laufenden Nummer 5.

A - RS/6000 AIX Systemdokumentation:

- [1] IBM RS/6000 AIX, User's Guide
IBM SC23-
- [2] IBM RS/6000 AIX, Editing Concepts and Procedures,
IBM SC23-2212
- [3] IBM RS/6000 AIX, Commands Reference, Vol. 1-3,
IBM SC23-2199
- [4] IBM RS/6000 AIX, User Interface Programming Concepts,
IBM SC23-2209 (u.a. curses Bibliothek)
- [5] IBM RS/6000 AIX, Calls and Subroutine Reference, Vol. 1-6,
IBM SC23-2198 (u.a. BLAS)
- [6] IBM RS/6000 AIX, General Programming Concepts,
IBM SC23-2205
- [7] RS/6000 AIX, X-Windows User's Guide,
IBM SC23-2017
- [8] RS/6000 AIX, TCP/IP User's Guide,
IBM SC23-2309 (u.a. ftp, telnet)
- [9] RS/6000 AIX, XL C Language Reference,
IBM SC09-1260
- [10] RS/6000 AIX, XL C User's Guide,
IBM SC09-1259
- [11] RS/6000 AIX, XL Fortran Language Reference,
IBM SC09-1258
- [12] RS/6000 AIX, XL Fortran User's Guide,
IBM SC09-1257
- [13] RS/6000 AIX, XL Fortran and C
Optimization and Tuning Guide
- [14] RS/6000 AIX, ESSL Guide and Reference,
Vol. 1-3
IBM ???
- [15] RS/6000 AIX, OSL,
IBM ???
- [16] RS/6000 AIX, OSL GUI,
IBM ???
- [17] RS/6000 AIX, C++ User's Guide,
IBM ???

- [18] RS/6000 AIX, C++ Reference Guide,
IBM ???
- [19] RS/6000 AIX, C++ Class Library Reference,
IBM ???
- [20] RS/6000 AIX, Data Explorer
IBM ???

B - Weiterführende UNIX Literatur:

- [1] Managing Projects with make,
A. Oram, S. Talbott,
O'Reilly & Ass., 1991
- [2] Learning the vi Editor,
L. Lamb,
O'Reilly & Ass., 1990
- [3] Programming with curses,
Strang,
O'Reilly & Ass., 1991
- [5] Einführung in UNIX,
J. Gulbins,
Springer, 1985
- [6] Der UNIX Werkzeugkasten,
B. Kernighan, R. Pike,
Hanser-Verlag, 1987

C - Dokumentation zu NAG Produkten:

- [1] NAG Fortran Library Introductory Guide,
Mark 14,
NAG Ltd.
- [2] NAG Fortran Library Reference Manual,
Mark 15, Vol. 1-8
NAG Ltd.
- [4] NAG Graphics Library Concise Reference,
Mark 3,
NAG Ltd.
- [5] NAG Graphics Library Reference Manual,
Mark 3, Vol. 1-3
NAG Ltd.
- [6] NAG Fortran 90 Compiler, User's Guide
Version 1.2
NAG Ltd.

D - Weiterführende Dokumentation zu X und OSF/Motif:

- [1] The X Window System Series:
Vol 1: XLib Programming Manual
Vol 2: XLib Reference Manual

Vol 3: X Window System User's Guide
Vol 4: X Toolkit Intrinsic Programming Manual
Vol 5: X Toolkit Intrinsic Reference Manual
Vol 6: Motif Programming Manual
alle: O'Reilly & Ass.

E - Dokumentation zu weiteren Programmen und Bibliotheken:

- [1] CPLEX User's Guide
- [2] Maple
LanguageReference
Springer Verlag, 199x
- [3] Maple
Library Reference
Springer Verlag, 199x
- [4] Mathematica - A System for Doing Mathematics by Computer
Steven Wolfram

F - Broschüren des Rechenzentrums:

- [1] Software unter UNIX
- [2] Leitfaden für Benutzer des Rechenzentrums