References

The references collected here are those of general usefulness, usually cited in more than one section of this book. More specialized sources, usually cited in a single section, are not repeated here.

We first list a small number of books that form the nucleus of a recommended personal reference collection on numerical methods, numerical analysis, and closely related subjects. These are the books that we like to have within easy reach.

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York)

Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America)

Ames, W.F. 1977, *Numerical Methods for Partial Differential Equations*, 2nd ed. (New York: Academic Press)

Bratley, P., Fox, B.L., and Schrage, E.L. 1983, *A Guide to Simulation* (New York: Springer-Verlag)

Dahlquist, G., and Bjorck, A. 1974, *Numerical Methods* (Englewood Cliffs, NJ: Prentice-Hall)

Delves, L.M., and Mohamed, J.L. 1985, *Computational Methods for Integral Equations* (Cambridge, U.K.: Cambridge University Press)

Dennis, J.E., and Schnabel, R.B. 1983, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (Englewood Cliffs, NJ: Prentice-Hall)

Gill, P.E., Murray, W., and Wright, M.H. 1991, *Numerical Linear Algebra and Optimization*, vol. 1 (Redwood City, CA: Addison-Wesley)

Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press)

Oppenheim, A.V., and Schafer, R.W. 1989, *Discrete-Time Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall)

Ralston, A., and Rabinowitz, P. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill)

Sedgewick, R. 1988, *Algorithms*, 2nd ed. (Reading, MA: Addison-Wesley)

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag)

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag)

We next list the larger collection of books, which, in our view, should be included in any serious research library on computing, numerical methods, or analysis.

Bevington, P.R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill)

Bloomfield, P. 1976, *Fourier Analysis of Time Series – An Introduction* (New York: Wiley)

Bowers, R.L., and Wilson, J.R. 1991, *Numerical Modeling in Applied Physics and Astrophysics* (Boston: Jones & Bartlett)

Brent, R.P. 1973, *Algorithms for Minimization without Derivatives* (Englewood Cliffs, NJ: Prentice-Hall)

Brigham, E.O. 1974, *The Fast Fourier Transform* (Englewood Cliffs, NJ: Prentice-Hall)

Brownlee, K.A. 1965, *Statistical Theory and Methodology*, 2nd ed. (New York: Wiley)

Bunch, J.R., and Rose, D.J. (eds.) 1976, *Sparse Matrix Computations* (New York: Academic Press)

Canuto, C., Hussaini, M.Y., Quarteroni, A., and Zang, T.A. 1988, *Spectral Methods in Fluid Dynamics* (New York: Springer-Verlag)

Carnahan, B., Luther, H.A., and Wilkes, J.O. 1969, *Applied Numerical Methods* (New York: Wiley)

Champeney, D.C. 1973, *Fourier Transforms and Their Physical Applications* (New York: Academic Press)

Childers, D.G. (ed.) 1978, *Modern Spectrum Analysis* (New York: IEEE Press)

Cooper, L., and Steinberg, D. 1970, *Introduction to Methods of Optimization* (Philadelphia: Saunders)

Dantzig, G.B. 1963, *Linear Programming and Extensions* (Princeton, NJ: Princeton University Press)

Devroye, L. 1986, *Non-Uniform Random Variate Generation* (New York: Springer-Verlag)

Dongarra, J.J., et al. 1979, *LINPACK User's Guide* (Philadelphia: S.I.A.M.)

Downie, N.M., and Heath, R.W. 1965, *Basic Statistical Methods*, 2nd ed. (New York: Harper & Row)

Duff, I.S., and Stewart, G.W. (eds.) 1979, *Sparse Matrix Proceedings 1978* (Philadelphia: S.I.A.M.)

Elliott, D.F., and Rao, K.R. 1982, *Fast Transforms: Algorithms, Analyses, Applications* (New York: Academic Press)

Fike, C.T. 1968, *Computer Evaluation of Mathematical Functions* (Englewood Cliffs, NJ: Prentice-Hall)

Forsythe, G.E., Malcolm, M.A., and Moler, C.B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, NJ: Prentice-Hall)

Forsythe, G.E., and Moler, C.B. 1967, *Computer Solution of Linear Algebraic Systems* (Englewood Cliffs, NJ: Prentice-Hall)

Gass, S.T. 1969, *Linear Programming*, 3rd ed. (New York: McGraw-Hill)

Gear, C.W. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, NJ: Prentice-Hall)

Goodwin, E.T. (ed.) 1961, *Modern Computing Methods*, 2nd ed. (New York: Philosophical Library)

Gottlieb, D. and Orszag, S.A. 1977, *Numerical Analysis of Spectral Methods: Theory and Applications* (Philadelphia: S.I.A.M.)

Hackbusch, W. 1985, *Multi-Grid Methods and Applications* (New York: Springer-Verlag)

Hamming, R.W. 1962, *Numerical Methods for Engineers and Scientists*; reprinted 1986 (New York: Dover)

Hart, J.F., et al. 1968, *Computer Approximations* (New York: Wiley)

Hastings, C. 1955, *Approximations for Digital Computers* (Princeton: Princeton University Press)

Hildebrand, F.B. 1974, *Introduction to Numerical Analysis*, 2nd ed.; reprinted 1987 (New York: Dover)

Hoel, P.G. 1971, *Introduction to Mathematical Statistics*, 4th ed. (New York: Wiley)

Horn, R.A., and Johnson, C.R. 1985, *Matrix Analysis* (Cambridge: Cambridge University Press)

Householder, A.S. 1970, *The Numerical Treatment of a Single Nonlinear Equation* (New York: McGraw-Hill)

Huber, P.J. 1981, *Robust Statistics* (New York: Wiley)

Isaacson, E., and Keller, H.B. 1966, *Analysis of Numerical Methods* (New York: Wiley)

Jacobs, D.A.H. (ed.) 1977, *The State of the Art in Numerical Analysis* (London: Academic Press)

Johnson, L.W., and Riess, R.D. 1982, *Numerical Analysis*, 2nd ed. (Reading, MA: Addison-Wesley)

Kahaner, D., Moler, C., and Nash, S. 1989, *Numerical Methods and Software* (Englewood Cliffs, NJ: Prentice Hall)

Keller, H.B. 1968, *Numerical Methods for Two-Point Boundary-Value Problems* (Waltham, MA: Blaisdell)

Knuth, D.E. 1968, *Fundamental Algorithms*, vol. 1 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley)

Knuth, D.E. 1981, *Seminumerical Algorithms*, 2nd ed., vol. 2 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley)

Knuth, D.E. 1973, *Sorting and Searching*, vol. 3 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley)

Koonin, S.E., and Meredith, D.C. 1990, *Computational Physics, Fortran Version* (Redwood City, CA: Addison-Wesley)

Kuenzi, H.P., Tzschach, H.G., and Zehnder, C.A. 1971, *Numerical Methods of Mathematical Optimization* (New York: Academic Press)

Lanczos, C. 1956, *Applied Analysis*; reprinted 1988 (New York: Dover)

Land, A.H., and Powell, S. 1973, *Fortran Codes for Mathematical Programming* (London: Wiley-Interscience)

Lawson, C.L., and Hanson, R. 1974, *Solving Least Squares Problems* (Englewood Cliffs, NJ: Prentice-Hall)

Lehmann, E.L. 1975, *Nonparametrics: Statistical Methods Based on Ranks* (San Francisco: Holden-Day)

Luke, Y.L. 1975, *Mathematical Functions and Their Approximations* (New York: Academic Press)

Magnus, W., and Oberhettinger, F. 1949, *Formulas and Theorems for the Functions of Mathematical Physics* (New York: Chelsea)

Martin, B.R. 1971, *Statistics for Physicists* (New York: Academic Press)

Mathews, J., and Walker, R.L. 1970, *Mathematical Methods of Physics*, 2nd ed. (Reading, MA: W.A. Benjamin/Addison-Wesley)

von Mises, R. 1964, *Mathematical Theory of Probability and Statistics* (New York: Academic Press)

Murty, K.G. 1976, *Linear and Combinatorial Programming* (New York: Wiley)

Norusis, M.J. 1982, *SPSS Introductory Guide: Basic Statistics and Operations*; and 1985, *SPSS-X Advanced Statistics Guide* (New York: McGraw-Hill)

Nussbaumer, H.J. 1982, *Fast Fourier Transform and Convolution Algorithms* (New York: Springer-Verlag)

Ortega, J., and Rheinboldt, W. 1970, *Iterative Solution of Nonlinear Equations in Several Variables* (New York: Academic Press)

Ostrowski, A.M. 1966, *Solutions of Equations and Systems of Equations*, 2nd ed. (New York: Academic Press)

Polak, E. 1971, *Computational Methods in Optimization* (New York: Academic Press)

Rice, J.R. 1983, *Numerical Methods, Software, and Analysis* (New York: McGraw-Hill)

Richtmyer, R.D., and Morton, K.W. 1967, *Difference Methods for Initial Value Problems*, 2nd ed. (New York: Wiley-Interscience)

Roache, P.J. 1976, *Computational Fluid Dynamics* (Albuquerque: Hermosa)

Robinson, E.A., and Treitel, S. 1980, *Geophysical Signal Analysis* (Englewood Cliffs, NJ: Prentice-Hall)

Smith, B.T., et al. 1976, *Matrix Eigensystem Routines — EISPACK Guide*, 2nd ed., vol. 6 of Lecture Notes in Computer Science (New York: Springer-Verlag)

Stuart, A., and Ord, J.K. 1987, *Kendall's Advanced Theory of Statistics*, 5th ed. (London: Griffin and Co.) [previous eds. published as Kendall, M., and Stuart, A., *The Advanced Theory of Statistics*]

Tewarson, R.P. 1973, *Sparse Matrices* (New York: Academic Press)

Westlake, J.R. 1968, *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations* (New York: Wiley)

Wilkinson, J.H. 1965, *The Algebraic Eigenvalue Problem* (New York: Oxford University Press)

Young, D.M., and Gregory, R.T. 1973, *A Survey of Numerical Mathematics*, 2 vols.; reprinted 1988 (New York: Dover)

Appendix A: Table of Prototype Declarations

We here list ANSI prototype declarations for all the routines in *Numerical Recipes in C*. If, as is preferred, you are using a compiler that implements the ANSI standard, then you should `#include` this listing (or the relevant lines thereof) in each separately compiled source file that contains or references any routine from this book. That will alert your compiler to the fact that our routines do not generally expect argument conversions, and also allow your compiler to point out possible errors in your invocation of our routines.

On the diskette, this Appendix is in the file `nr.h`. An important point about the file on the diskette is that it contains *both* ANSI C and "traditional" or "K&R" style declarations. The ANSI forms are invoked if any of the following macros are defined: `__STDC__`, `ANSI`, or `NRANSI`. (The purpose of the last one is to give you an invocation that does not conflict with other possible uses of the first two names.) If you do have an ANSI compiler it is *essential* that you invoke it with one or more of these macros defined. The typical means for doing so is to include a switch like "`-DANSI`" on the compiler command line.

If you have a "traditional" or "K&R" `C` compiler, then the above discussion does not apply to you: you do not need the ANSI header file listed here. If you have the diskette `nr.h` file, you will probably find it helpful to `#include nr.h` anyway, *without* setting any of the macros `__STDC__`, `ANSI`, or `NRANSI`. That will at least alert your compiler to the returned value types of our routines. You should of course be sure to use the K&R versions of our programs, as included (along with the primary ANSI versions) on the *Numerical Recipes C Diskette*. Your compiler will do the "usual argument conversions" whether you like it or not, but it will also *un*do them upon entering routines whose arguments have been declared differently from the usual conversions.

Here is a listing of the file `nr.h`:

```
#ifndef _NR_H_
#define _NR_H_

#ifndef _FCOMPLEX_DECLARE_T_
typedef struct FCOMPLEX {float r,i;} fcomplex;
#define _FCOMPLEX_DECLARE_T_
#endif /* _FCOMPLEX_DECLARE_T_ */

#ifndef _ARITHCODE_DECLARE_T_
typedef struct {
```

```
    unsigned long *ilob,*iupb,*ncumfq,jdif,nc,minint,nch,ncum,nrad;
} arithcode;
#define _ARITHCODE_DECLARE_T_
#endif /* _ARITHCODE_DECLARE_T_ */

#ifndef _HUFFCODE_DECLARE_T_
typedef struct {
    unsigned long *icod,*ncod,*left,*right,nch,nodemax;
} huffcode;
#define _HUFFCODE_DECLARE_T_
#endif /* _HUFFCODE_DECLARE_T_ */

#include <stdio.h>

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

void addint(double **uf, double **uc, double **res, int nf);
void airy(float x, float *ai, float *bi, float *aip, float *bip);
void amebsa(float **p, float y[], int ndim, float pb[],    float *yb,
    float ftol, float (*funk)(float []), int *iter, float temptr);
void amoeba(float **p, float y[], int ndim, float ftol,
    float (*funk)(float []), int *iter);
float amotry(float **p, float y[], float psum[], int ndim,
    float (*funk)(float []), int ihi, float fac);
float amotsa(float **p, float y[], float psum[], int ndim, float pb[],
    float *yb, float (*funk)(float []), int ihi, float *yhi, float fac);
void anneal(float x[], float y[], int iorder[], int ncity);
double anorm2(double **a, int n);
void arcmak(unsigned long nfreq[], unsigned long nchh, unsigned long nradd,
    arithcode *acode);
void arcode(unsigned long *ich, unsigned char **codep, unsigned long *lcode,
    unsigned long *lcd, int isign, arithcode *acode);
void arcsum(unsigned long iin[], unsigned long iout[], unsigned long ja,
    int nwk, unsigned long nrad, unsigned long nc);
void asolve(unsigned long n, double b[], double x[], int itrnsp);
void atimes(unsigned long n, double x[], double r[], int itrnsp);
void avevar(float data[], unsigned long n, float *ave, float *var);
void balanc(float **a, int n);
void banbks(float **a, unsigned long n, int m1, int m2, float **al,
    unsigned long indx[], float b[]);
void bandec(float **a, unsigned long n, int m1, int m2, float **al,
    unsigned long indx[], float *d);
void banmul(float **a, unsigned long n, int m1, int m2, float x[], float b[]);
void bcucof(float y[], float y1[], float y2[], float y12[], float d1,
    float d2, float **c);
void bcuint(float y[], float y1[], float y2[], float y12[],
    float x1l, float x1u, float x2l, float x2u, float x1,
    float x2, float *ansy, float *ansy1, float *ansy2);
void beschb(double x, double *gam1, double *gam2, double *gampl,
    double *gammi);
float bessi(int n, float x);
float bessi0(float x);
float bessi1(float x);
void bessik(float x, float xnu, float *ri, float *rk, float *rip,
    float *rkp);
float bessj(int n, float x);
float bessj0(float x);
float bessj1(float x);
void bessjy(float x, float xnu, float *rj, float *ry, float *rjp,
    float *ryp);
float bessk(int n, float x);
float bessk0(float x);
float bessk1(float x);
float bessy(int n, float x);
```

```
float bessy0(float x);
float bessy1(float x);
float beta(float z, float w);
float betacf(float a, float b, float x);
float betai(float a, float b, float x);
float bico(int n, int k);
void bksub(int ne, int nb, int jf, int k1, int k2, float ***c);
float bnldev(float pp, int n, long *idum);
float brent(float ax, float bx, float cx,
    float (*f)(float), float tol, float *xmin);
void broydn(float x[], int n, int *check,
    void (*vecfunc)(int, float [], float []));
void bsstep(float y[], float dydx[], int nv, float *xx, float htry,
    float eps, float yscal[], float *hdid, float *hnext,
    void (*derivs)(float, float [], float []));
void caldat(long julian, int *mm, int *id, int *iyyy);
void chder(float a, float b, float c[], float cder[], int n);
float chebev(float a, float b, float c[], int m, float x);
void chebft(float a, float b, float c[], int n, float (*func)(float));
void chebpc(float c[], float d[], int n);
void chint(float a, float b, float c[], float cint[], int n);
float chixy(float bang);
void choldc(float **a, int n, float p[]);
void cholsl(float **a, int n, float p[], float b[], float x[]);
void chsone(float bins[], float ebins[], int nbins, int knstrn,
    float *df, float *chsq, float *prob);
void chstwo(float bins1[], float bins2[], int nbins, int knstrn,
    float *df, float *chsq, float *prob);
void cisi(float x, float *ci, float *si);
void cntab1(int **nn, int ni, int nj, float *chisq,
    float *df, float *prob, float *cramrv, float *ccc);
void cntab2(int **nn, int ni, int nj, float *h, float *hx, float *hy,
    float *hygx, float *hxgy, float *uygx, float *uxgy, float *uxy);
void convlv(float data[], unsigned long n, float respns[], unsigned long m,
    int isign, float ans[]);
void copy(double **aout, double **ain, int n);
void correl(float data1[], float data2[], unsigned long n, float ans[]);
void cosft(float y[], int n, int isign);
void cosft1(float y[], int n);
void cosft2(float y[], int n, int isign);
void covsrt(float **covar, int ma, int ia[], int mfit);
void crank(unsigned long n, float w[], float *s);
void cyclic(float a[], float b[], float c[], float alpha, float beta,
    float r[], float x[], unsigned long n);
void daub4(float a[], unsigned long n, int isign);
float dawson(float x);
float dbrent(float ax, float bx, float cx,
    float (*f)(float), float (*df)(float), float tol, float *xmin);
void ddpoly(float c[], int nc, float x, float pd[], int nd);
int decchk(char string[], int n, char *ch);
void derivs(float x, float y[], float dydx[]);
float df1dim(float x);
void dfour1(double data[], unsigned long nn, int isign);
void dfpmin(float p[], int n, float gtol, int *iter, float *fret,
    float (*func)(float []), void (*dfunc)(float [], float []));
float dfridr(float (*func)(float), float x, float h, float *err);
void dftcor(float w, float delta, float a, float b, float endpts[],
    float *corre, float *corim, float *corfac);
void dftint(float (*func)(float), float a, float b, float w,
    float *cosint, float *sinint);
void difeq(int k, int k1, int k2, int jsf, int is1, int isf,
    int indexv[], int ne, float **s, float **y);
void dlinmin(float p[], float xi[], int n, float *fret,
    float (*func)(float []), void (*dfunc)(float [], float[]));
```

```
double dpythag(double a, double b);
void drealft(double data[], unsigned long n, int isign);
void dsprsax(double sa[], unsigned long ija[], double x[], double b[],
    unsigned long n);
void dsprstx(double sa[], unsigned long ija[], double x[], double b[],
    unsigned long n);
void dsvbksb(double **u, double w[], double **v, int m, int n, double b[],
    double x[]);
void dsvdcmp(double **a, int m, int n, double w[], double **v);
void eclass(int nf[], int n, int lista[], int listb[], int m);
void eclazz(int nf[], int n, int (*equiv)(int, int));
float ei(float x);
void eigsrt(float d[], float **v, int n);
float elle(float phi, float ak);
float ellf(float phi, float ak);
float ellpi(float phi, float en, float ak);
void elmhes(float **a, int n);
float erfcc(float x);
float erff(float x);
float erffc(float x);
void eulsum(float *sum, float term, int jterm, float wksp[]);
float evlmem(float fdt, float d[], int m, float xms);
float expdev(long *idum);
float expint(int n, float x);
float f1(float x);
float f1dim(float x);
float f2(float y);
float f3(float z);
float factln(int n);
float factrl(int n);
void fasper(float x[], float y[], unsigned long n, float ofac, float hifac,
    float wk1[], float wk2[], unsigned long nwk, unsigned long *nout,
    unsigned long *jmax, float *prob);
void fdjac(int n, float x[], float fvec[], float **df,
    void (*vecfunc)(int, float [], float []));
void fgauss(float x, float a[], float *y, float dyda[], int na);
void fill0(double **u, int n);
void fit(float x[], float y[], int ndata, float sig[], int mwt,
    float *a, float *b, float *siga, float *sigb, float *chi2, float *q);
void fitexy(float x[], float y[], int ndat, float sigx[], float sigy[],
    float *a, float *b, float *siga, float *sigb, float *chi2, float *q);
void fixrts(float d[], int m);
void fleg(float x, float pl[], int nl);
void flmoon(int n, int nph, long *jd, float *frac);
float fmin(float x[]);
void four1(float data[], unsigned long nn, int isign);
void fourew(FILE *file[5], int *na, int *nb, int *nc, int *nd);
void fourfs(FILE *file[5], unsigned long nn[], int ndim, int isign);
void fourn(float data[], unsigned long nn[], int ndim, int isign);
void fpoly(float x, float p[], int np);
void fred2(int n, float a, float b, float t[], float f[], float w[],
    float (*g)(float), float (*ak)(float, float));
float fredin(float x, int n, float a, float b, float t[], float f[], float w[],
    float (*g)(float), float (*ak)(float, float));
void frenel(float x, float *s, float *c);
void frprmn(float p[], int n, float ftol, int *iter, float *fret,
    float (*func)(float []), void (*dfunc)(float [], float []));
void ftest(float data1[], unsigned long n1, float data2[], unsigned long n2,
    float *f, float *prob);
float gamdev(int ia, long *idum);
float gammln(float xx);
float gammp(float a, float x);
float gammq(float a, float x);
float gasdev(long *idum);
```

```
void gaucof(int n, float a[], float b[], float amu0, float x[], float w[]);
void gauher(float x[], float w[], int n);
void gaujac(float x[], float w[], int n, float alf, float bet);
void gaulag(float x[], float w[], int n, float alf);
void gauleg(float x1, float x2, float x[], float w[], int n);
void gaussj(float **a, int n, float **b, int m);
void gcf(float *gammcf, float a, float x, float *gln);
float golden(float ax, float bx, float cx, float (*f)(float), float tol,
    float *xmin);
void gser(float *gamser, float a, float x, float *gln);
void hpsel(unsigned long m, unsigned long n, float arr[], float heap[]);
void hpsort(unsigned long n, float ra[]);
void hqr(float **a, int n, float wr[], float wi[]);
void hufapp(unsigned long index[], unsigned long nprob[], unsigned long n,
    unsigned long i);
void hufdec(unsigned long *ich, unsigned char *code, unsigned long lcode,
    unsigned long *nb, huffcode *hcode);
void hufenc(unsigned long ich, unsigned char **codep, unsigned long *lcode,
    unsigned long *nb, huffcode *hcode);
void hufmak(unsigned long nfreq[], unsigned long nchin, unsigned long *ilong,
    unsigned long *nlong, huffcode *hcode);
void hunt(float xx[], unsigned long n, float x, unsigned long *jlo);
void hypdrv(float s, float yy[], float dyyds[]);
fcomplex hypgeo(fcomplex a, fcomplex b, fcomplex c, fcomplex z);
void hypser(fcomplex a, fcomplex b, fcomplex c, fcomplex z,
    fcomplex *series, fcomplex *deriv);
unsigned short icrc(unsigned short crc, unsigned char *bufptr,
    unsigned long len, short jinit, int jrev);
unsigned short icrc1(unsigned short crc, unsigned char onech);
unsigned long igray(unsigned long n, int is);
void iindexx(unsigned long n, long arr[], unsigned long indx[]);
void indexx(unsigned long n, float arr[], unsigned long indx[]);
void interp(double **uf, double **uc, int nf);
int irbit1(unsigned long *iseed);
int irbit2(unsigned long *iseed);
void jacobi(float **a, int n, float d[], float **v, int *nrot);
void jacobn(float x, float y[], float dfdx[], float **dfdy, int n);
long julday(int mm, int id, int iyyy);
void kendl1(float data1[], float data2[], unsigned long n, float *tau, float *z,
    float *prob);
void kendl2(float **tab, int i, int j, float *tau, float *z, float *prob);
void kermom(double w[], double y, int m);
void ks2d1s(float x1[], float y1[], unsigned long n1,
    void (*quadvl)(float, float, float *, float *, float *, float *),
    float *d1, float *prob);
void ks2d2s(float x1[], float y1[], unsigned long n1, float x2[], float y2[],
    unsigned long n2, float *d, float *prob);
void ksone(float data[], unsigned long n, float (*func)(float), float *d,
    float *prob);
void kstwo(float data1[], unsigned long n1, float data2[], unsigned long n2,
    float *d, float *prob);
void laguer(fcomplex a[], int m, fcomplex *x, int *its);
void lfit(float x[], float y[], float sig[], int ndat, float a[], int ia[],
    int ma, float **covar, float *chisq, void (*funcs)(float, float [], int));
void linbcg(unsigned long n, double b[], double x[], int itol, double tol,
     int itmax, int *iter, double *err);
void linmin(float p[], float xi[], int n, float *fret,
    float (*func)(float []));
void lnsrch(int n, float xold[], float fold, float g[], float p[], float x[],
     float *f, float stpmax, int *check, float (*func)(float []));
void load(float x1, float v[], float y[]);
void load1(float x1, float v1[], float y[]);
void load2(float x2, float v2[], float y[]);
void locate(float xx[], unsigned long n, float x, unsigned long *j);
```

```
void lop(double **out, double **u, int n);
void lubksb(float **a, int n, int *indx, float b[]);
void ludcmp(float **a, int n, int *indx, float *d);
void machar(int *ibeta, int *it, int *irnd, int *ngrd,
    int *machep, int *negep, int *iexp, int *minexp, int *maxexp,
    float *eps, float *epsneg, float *xmin, float *xmax);
void matadd(double **a, double **b, double **c, int n);
void matsub(double **a, double **b, double **c, int n);
void medfit(float x[], float y[], int ndata, float *a, float *b, float *abdev);
void memcof(float data[], int n, int m, float *xms, float d[]);
int metrop(float de, float t);
void mgfas(double **u, int n, int maxcyc);
void mglin(double **u, int n, int ncycle);
float midexp(float (*funk)(float), float aa, float bb, int n);
float midinf(float (*funk)(float), float aa, float bb, int n);
float midpnt(float (*func)(float), float a, float b, int n);
float midsql(float (*funk)(float), float aa, float bb, int n);
float midsqu(float (*funk)(float), float aa, float bb, int n);
void miser(float (*func)(float []), float regn[], int ndim, unsigned long npts,
    float dith, float *ave, float *var);
void mmid(float y[], float dydx[], int nvar, float xs, float htot,
    int nstep, float yout[], void (*derivs)(float, float[], float[]));
void mnbrak(float *ax, float *bx, float *cx, float *fa, float *fb,
    float *fc, float (*func)(float));
void mnewt(int ntrial, float x[], int n, float tolx, float tolf);
void moment(float data[], int n, float *ave, float *adev, float *sdev,
    float *var, float *skew, float *curt);
void mp2dfr(unsigned char a[], unsigned char s[], int n, int *m);
void mpadd(unsigned char w[], unsigned char u[], unsigned char v[], int n);
void mpdiv(unsigned char q[], unsigned char r[], unsigned char u[],
    unsigned char v[], int n, int m);
void mpinv(unsigned char u[], unsigned char v[], int n, int m);
void mplsh(unsigned char u[], int n);
void mpmov(unsigned char u[], unsigned char v[], int n);
void mpmul(unsigned char w[], unsigned char u[], unsigned char v[], int n,
    int m);
void mpneg(unsigned char u[], int n);
void mppi(int n);
void mprove(float **a, float **alud, int n, int indx[], float b[],
    float x[]);
void mpsad(unsigned char w[], unsigned char u[], int n, int iv);
void mpsdv(unsigned char w[], unsigned char u[], int n, int iv, int *ir);
void mpsmu(unsigned char w[], unsigned char u[], int n, int iv);
void mpsqrt(unsigned char w[], unsigned char u[], unsigned char v[], int n,
    int m);
void mpsub(int *is, unsigned char w[], unsigned char u[], unsigned char v[],
    int n);
void mrqcof(float x[], float y[], float sig[], int ndata, float a[],
    int ia[], int ma, float **alpha, float beta[], float *chisq,
    void (*funcs)(float, float [], float *, float [], int));
void mrqmin(float x[], float y[], float sig[], int ndata, float a[],
    int ia[], int ma, float **covar, float **alpha, float *chisq,
    void (*funcs)(float, float [], float *, float [], int), float *alamda);
void newt(float x[], int n, int *check,
    void (*vecfunc)(int, float [], float []));
void odeint(float ystart[], int nvar, float x1, float x2,
    float eps, float h1, float hmin, int *nok, int *nbad,
    void (*derivs)(float, float [], float []),
    void (*rkqs)(float [], float [], int, float *, float, float,
    float [], float *, float *, void (*)(float, float [], float [])));
void orthog(int n, float anu[], float alpha[], float beta[], float a[],
    float b[]);
void pade(double cof[], int n, float *resid);
void pccheb(float d[], float c[], int n);
```

```
void pcshft(float a, float b, float d[], int n);
void pearsn(float x[], float y[], unsigned long n, float *r, float *prob,
    float *z);
void period(float x[], float y[], int n, float ofac, float hifac,
    float px[], float py[], int np, int *nout, int *jmax, float *prob);
void piksr2(int n, float arr[], float brr[]);
void piksrt(int n, float arr[]);
void pinvs(int ie1, int ie2, int je1, int jsf, int jc1, int k,
    float ***c, float **s);
float plgndr(int l, int m, float x);
float poidev(float xm, long *idum);
void polcoe(float x[], float y[], int n, float cof[]);
void polcof(float xa[], float ya[], int n, float cof[]);
void poldiv(float u[], int n, float v[], int nv, float q[], float r[]);
void polin2(float x1a[], float x2a[], float **ya, int m, int n,
    float x1, float x2, float *y, float *dy);
void polint(float xa[], float ya[], int n, float x, float *y, float *dy);
void powell(float p[], float **xi, int n, float ftol, int *iter, float *fret,
    float (*func)(float []));
void predic(float data[], int ndata, float d[], int m, float future[], int nfut);
float probks(float alam);
void psdes(unsigned long *lword, unsigned long *irword);
void pwt(float a[], unsigned long n, int isign);
void pwtset(int n);
float pythag(float a, float b);
void pzextr(int iest, float xest, float yest[], float yz[], float dy[],
    int nv);
float qgaus(float (*func)(float), float a, float b);
void qrdcmp(float **a, int n, float *c, float *d, int *sing);
float qromb(float (*func)(float), float a, float b);
float qromo(float (*func)(float), float a, float b,
    float (*choose)(float (*)(float), float, float, int));
void qroot(float p[], int n, float *b, float *c, float eps);
void qrsolv(float **a, int n, float c[], float d[], float b[]);
void qrupdt(float **r, float **qt, int n, float u[], float v[]);
float qsimp(float (*func)(float), float a, float b);
float qtrap(float (*func)(float), float a, float b);
float quad3d(float (*func)(float, float, float), float x1, float x2);
void quadct(float x, float y, float xx[], float yy[], unsigned long nn,
    float *fa, float *fb, float *fc, float *fd);
void quadmx(float **a, int n);
void quadvl(float x, float y, float *fa, float *fb, float *fc, float *fd);
float ran0(long *idum);
float ran1(long *idum);
float ran2(long *idum);
float ran3(long *idum);
float ran4(long *idum);
void rank(unsigned long n, unsigned long indx[], unsigned long irank[]);
void ranpt(float pt[], float regn[], int n);
void ratint(float xa[], float ya[], int n, float x, float *y, float *dy);
void ratlsq(double (*fn)(double), double a, double b, int mm, int kk,
    double cof[], double *dev);
double ratval(double x, double cof[], int mm, int kk);
float rc(float x, float y);
float rd(float x, float y, float z);
void realft(float data[], unsigned long n, int isign);
void rebin(float rc, int nd, float r[], float xin[], float xi[]);
void red(int iz1, int iz2, int jz1, int jz2, int jm1, int jm2, int jmf,
    int ic1, int jc1, int jcf, int kc, float ***c, float **s);
void relax(double **u, double **rhs, int n);
void relax2(double **u, double **rhs, int n);
void resid(double **res, double **u, double **rhs, int n);
float revcst(float x[], float y[], int iorder[], int ncity, int n[]);
void reverse(int iorder[], int ncity, int n[]);
```

```
float rf(float x, float y, float z);
float rj(float x, float y, float z, float p);
void rk4(float y[], float dydx[], int n, float x, float h, float yout[],
    void (*derivs)(float, float [], float []));
void rkck(float y[], float dydx[], int n, float x, float h,
    float yout[], float yerr[], void (*derivs)(float, float [], float []));
void rkdumb(float vstart[], int nvar, float x1, float x2, int nstep,
    void (*derivs)(float, float [], float []));
void rkqs(float y[], float dydx[], int n, float *x,
    float htry, float eps, float yscal[], float *hdid, float *hnext,
    void (*derivs)(float, float [], float []));
void rlft3(float ***data, float **speq, unsigned long nn1,
    unsigned long nn2, unsigned long nn3, int isign);
float rofunc(float b);
void rotate(float **r, float **qt, int n, int i, float a, float b);
void rsolv(float **a, int n, float d[], float b[]);
void rstrct(double **uc, double **uf, int nc);
float rtbis(float (*func)(float), float x1, float x2, float xacc);
float rtflsp(float (*func)(float), float x1, float x2, float xacc);
float rtnewt(void (*funcd)(float, float *, float *), float x1, float x2,
    float xacc);
float rtsafe(void (*funcd)(float, float *, float *), float x1, float x2,
    float xacc);
float rtsec(float (*func)(float), float x1, float x2, float xacc);
void rzextr(int iest, float xest, float yest[], float yz[], float dy[], int nv);
void savgol(float c[], int np, int nl, int nr, int ld, int m);
void score(float xf, float y[], float f[]);
void scrsho(float (*fx)(float));
float select(unsigned long k, unsigned long n, float arr[]);
float selip(unsigned long k, unsigned long n, float arr[]);
void shell(unsigned long n, float a[]);
void shoot(int n, float v[], float f[]);
void shootf(int n, float v[], float f[]);
void simp1(float **a, int mm, int ll[], int nll, int iabf, int *kp,
    float *bmax);
void simp2(float **a, int m, int n, int *ip, int kp);
void simp3(float **a, int i1, int k1, int ip, int kp);
void simplx(float **a, int m, int n, int m1, int m2, int m3, int *icase,
    int izrov[], int iposv[]);
void simpr(float y[], float dydx[], float dfdx[], float **dfdy,
    int n, float xs, float htot, int nstep, float yout[],
    void (*derivs)(float, float [], float []));
void sinft(float y[], int n);
void slvsm2(double **u, double **rhs);
void slvsml(double **u, double **rhs);
void sncndn(float uu, float emmc, float *sn, float *cn, float *dn);
double snrm(unsigned long n, double sx[], int itol);
void sobseq(int *n, float x[]);
void solvde(int itmax, float conv, float slowc, float scalv[],
    int indexv[], int ne, int nb, int m, float **y, float ***c, float **s);
void sor(double **a, double **b, double **c, double **d, double **e,
    double **f, double **u, int jmax, double rjac);
void sort(unsigned long n, float arr[]);
void sort2(unsigned long n, float arr[], float brr[]);
void sort3(unsigned long n, float ra[], float rb[], float rc[]);
void spctrm(FILE *fp, float p[], int m, int k, int ovrlap);
void spear(float data1[], float data2[], unsigned long n, float *d, float *zd,
    float *probd, float *rs, float *probrs);
void sphbes(int n, float x, float *sj, float *sy, float *sjp, float *syp);
void splie2(float x1a[], float x2a[], float **ya, int m, int n, float **y2a);
void splin2(float x1a[], float x2a[], float **ya, float **y2a, int m, int n,
    float x1, float x2, float *y);
void spline(float x[], float y[], int n, float yp1, float ypn, float y2[]);
void splint(float xa[], float ya[], float y2a[], int n, float x, float *y);
```

```
void spread(float y, float yy[], unsigned long n, float x, int m);
void sprsax(float sa[], unsigned long ija[], float x[], float b[],
    unsigned long n);
void sprsin(float **a, int n, float thresh, unsigned long nmax, float sa[],
    unsigned long ija[]);
void sprspm(float sa[], unsigned long ija[], float sb[], unsigned long ijb[],
    float sc[], unsigned long ijc[]);
void sprstm(float sa[], unsigned long ija[], float sb[], unsigned long ijb[],
    float thresh, unsigned long nmax, float sc[], unsigned long ijc[]);
void sprstp(float sa[], unsigned long ija[], float sb[], unsigned long ijb[]);
void sprstx(float sa[], unsigned long ija[], float x[], float b[],
    unsigned long n);
void stifbs(float y[], float dydx[], int nv, float *xx,
    float htry, float eps, float yscal[], float *hdid, float *hnext,
    void (*derivs)(float, float [], float []));
void stiff(float y[], float dydx[], int n, float *x,
    float htry, float eps, float yscal[], float *hdid, float *hnext,
    void (*derivs)(float, float [], float []));
void stoerm(float y[], float d2y[], int nv, float xs,
    float htot, int nstep, float yout[],
    void (*derivs)(float, float [], float []));
void svbksb(float **u, float w[], float **v, int m, int n, float b[],
    float x[]);
void svdcmp(float **a, int m, int n, float w[], float **v);
void svdfit(float x[], float y[], float sig[], int ndata, float a[],
    int ma, float **u, float **v, float w[], float *chisq,
    void (*funcs)(float, float [], int));
void svdvar(float **v, int ma, float w[], float **cvm);
void toeplz(float r[], float x[], float y[], int n);
void tptest(float data1[], float data2[], unsigned long n, float *t, float *prob);
void tqli(float d[], float e[], int n, float **z);
float trapzd(float (*func)(float), float a, float b, int n);
void tred2(float **a, int n, float d[], float e[]);
void tridag(float a[], float b[], float c[], float r[], float u[],
    unsigned long n);
float trncst(float x[], float y[], int iorder[], int ncity, int n[]);
void trnspt(int iorder[], int ncity, int n[]);
void ttest(float data1[], unsigned long n1, float data2[], unsigned long n2,
    float *t, float *prob);
void tutest(float data1[], unsigned long n1, float data2[], unsigned long n2,
    float *t, float *prob);
void twofft(float data1[], float data2[], float fft1[], float fft2[],
    unsigned long n);
void vander(double x[], double w[], double q[], int n);
void vegas(float regn[], int ndim, float (*fxn)(float [], float), int init,
    unsigned long ncall, int itmx, int nprn, float *tgral, float *sd,
    float *chi2a);
void voltra(int n, int m, float t0, float h, float *t, float **f,
    float (*g)(int, float), float (*ak)(int, int, float, float));
void wt1(float a[], unsigned long n, int isign,
    void (*wtstep)(float [], unsigned long, int));
void wtn(float a[], unsigned long nn[], int ndim, int isign,
    void (*wtstep)(float [], unsigned long, int));
void wwghts(float wghts[], int n, float h,
    void (*kermom)(double [], double ,int));
int zbrac(float (*func)(float), float *x1, float *x2);
void zbrak(float (*fx)(float), float x1, float x2, int n, float xb1[],
    float xb2[], int *nb);
float zbrent(float (*func)(float), float x1, float x2, float tol);
void zrhqr(float a[], int m, float rtr[], float rti[]);
float zriddr(float (*func)(float), float x1, float x2, float xacc);
void zroots(fcomplex a[], int m, fcomplex roots[], int polish);

#else /* ANSI */
```

```
/* traditional - K&R */

void addint();
void airy();
```

Rest of traditional declarations are here on the diskette.

```
#endif /* ANSI */

#endif /* _NR_H_ */
```

Appendix B: Utility Routines

Non-Copyright Notice: This Appendix and its utility routines are herewith placed into the public domain. Anyone may copy them freely for any purpose. We of course accept no liability whatsoever for any such use.

The routines listed below are used by many of the Recipes in this book. The first routine, `nrerror`, is invoked to terminate program execution — with an appropriate message — when a fatal error is encountered. The other routines are used to allocate and deallocate memory for vectors and matrices, as explained in detail in §1.2. *All* memory allocation and deallocation in *Numerical Recipes in C* is done using these routines as intermediaries. Therefore, if you want to allocate memory in some different way, you need change only these routines, not the programs themselves.

On the diskette, these routines are in the file `nrutil.c`. Also listed here is the header file `nrutil.h`. Besides the declarations for the routines in `nrutil.h`, it contains several macros that are used throughout the book. The ideas behind some of these macros were described in §1.2.

Memory Allocation: Advanced Topics

Two issues regarding vector and matrix memory allocation are worth discussing here: first, a "back-door" compatibility between our matrices and those used by other C software; second, the question of whether our pointer arithmetic for unit-offset vectors and matrices violates the ANSI C standard.

1. Back-door compatibility. As explained in §1.2, we *always* allocate matrices via the scheme "pointer to an array of pointers to rows" (shown in Figure 1.2.1). With this scheme, once storage for a matrix is allocated, the top level pointer (the "name" of the matrix) can be used to store a matrix of any size whose dimensions are less than or equal to the size of the dimensions allocated. The only time that the allocated size ever again becomes relevant is when the matrix storage is finally deallocated.

In this scheme, the rows of a matrix need not be stored in physically contiguous storage; each row has its own pointer addressing it (see Figure 1.2.1). However, the allocation routines printed below *in fact* (and by design) allocate an entire matrix as one contiguous block, via a single call to `malloc()`. This "back-door" fact allows you to use matrices created with our routines directly with other software: The address of the first element in a matrix `**a` (usually `&a[1][1]` if `a` has been allocated with a statement like `a=matrix(1,m,1,n);` but possibly `&a[0][0]` if `a` was created with zero-offsets) is guaranteed to point to the beginning of a contiguous block containing the full physical matrix, stored by rows. This address can be used as an argument to other software. (Note that this other software will generally also need to know the physical size of the matrix, or at least the number of columns.)

2. Unit-offset vectors. In §1.2, we described how a unit-offset vector `bb[1..4]` could be obtained from a zero-offset array `b[0..3]` by the pointer arithmetic operation `bb=b-1`. Since `bb` points to one location before `b`, `bb[1]` addresses the same element as `b[0]`, and so on.

Strictly speaking, this scheme is not blessed by the ANSI C standard. The problem is *not* the fact that `b-1` points to unallocated storage: location `b-1` will never be referenced without an increment back into the allocated region. Rather, the problem is that it might happen in rare cases (and probably only on a segmented machine) that the expression `b-1` has *no representation at all*. If this occurs, then there is no guarantee that the relation `b=(b-1)+1` is satisfied.

In practice, this is much less of a problem than one might think. We are not aware of any compiler or machine on which `b=(b-`n`)+`n fails to be true for small integer n. Even on a segmented machine, what typically happens is that the compiler stores *some* (perhaps illegal) representation of `b-`n, and that `b` is recovered when n is added back to this representation. The memory allocation routines in the First Edition of *Numerical Recipes in C*, in wide use since 1988, all have this "problem", and we have had not even a single report of their failure in this respect (notwithstanding the many readers who have told us that *theoretically* it could fail). We have also communicated to standards bodies the desirability of blessing "`b=(b-`n`)+`n" (at least for some range of n, say n representable as type `short`) in a future standard, since there would seem to be no conflict with existing compilers in doing so.

Despite the absence of any experimental (as opposed to theoretical) problem, we have taken some steps in this edition to make our vector and matrix allocation routines more ANSI compliant. In the listing that follows, the parameter `NR_END` is used as a number of extra storage locations allocated *at the beginning* of every vector or matrix block, simply for the purpose of making offset pointer references guaranteed-representable. We set `NR_END` to a default value of 1. This has the effect of making all *unit-offset* allocations (e.g., `b=vector(1,7);` or `a=matrix(1,128,1,128);`) be strictly ANSI compliant. With `NR_END` $= 1$, the number of storage locations wasted is fairly negligible. Allocations with offsets other than 1 (e.g., `b=vector(2,10)`) are still theoretically non-compliant, but are virtually unknown in our routines. If you need to make such allocations, you may wish to consider increasing the value of `NR_END` (noting that larger values increase the amount of wasted storage).

Here is a listing of `nrutil.h`:

```
#ifndef _NR_UTILS_H_
#define _NR_UTILS_H_

static float sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)

static double dsqrarg;
#define DSQR(a) ((dsqrarg=(a)) == 0.0 ? 0.0 : dsqrarg*dsqrarg)

static double dmaxarg1,dmaxarg2;
#define DMAX(a,b) (dmaxarg1=(a),dmaxarg2=(b),(dmaxarg1) > (dmaxarg2) ?\
(dmaxarg1) : (dmaxarg2))

static double dminarg1,dminarg2;
#define DMIN(a,b) (dminarg1=(a),dminarg2=(b),(dminarg1) < (dminarg2) ?\
(dminarg1) : (dminarg2))

static float maxarg1,maxarg2;
#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
(maxarg1) : (maxarg2))

static float minarg1,minarg2;
#define FMIN(a,b) (minarg1=(a),minarg2=(b),(minarg1) < (minarg2) ?\
(minarg1) : (minarg2))

static long lmaxarg1,lmaxarg2;
#define LMAX(a,b) (lmaxarg1=(a),lmaxarg2=(b),(lmaxarg1) > (lmaxarg2) ?\
(lmaxarg1) : (lmaxarg2))

static long lminarg1,lminarg2;
```

```
#define LMIN(a,b) (lminarg1=(a),lminarg2=(b),(lminarg1) < (lminarg2) ?\
(lminarg1) : (lminarg2))

static int imaxarg1,imaxarg2;
#define IMAX(a,b) (imaxarg1=(a),imaxarg2=(b),(imaxarg1) > (imaxarg2) ?\
(imaxarg1) : (imaxarg2))

static int iminarg1,iminarg2;
#define IMIN(a,b) (iminarg1=(a),iminarg2=(b),(iminarg1) < (iminarg2) ?\
(iminarg1) : (iminarg2))

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

void nrerror(char error_text[]);
float *vector(long nl, long nh);
int *ivector(long nl, long nh);
unsigned char *cvector(long nl, long nh);
unsigned long *lvector(long nl, long nh);
double *dvector(long nl, long nh);
float **matrix(long nrl, long nrh, long ncl, long nch);
double **dmatrix(long nrl, long nrh, long ncl, long nch);
int **imatrix(long nrl, long nrh, long ncl, long nch);
float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,
    long newrl, long newcl);
float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch);
float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh);
void free_vector(float *v, long nl, long nh);
void free_ivector(int *v, long nl, long nh);
void free_cvector(unsigned char *v, long nl, long nh);
void free_lvector(unsigned long *v, long nl, long nh);
void free_dvector(double *v, long nl, long nh);
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch);
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch);
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch);
void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch);
void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch);
void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
    long ndl, long ndh);

#else /* ANSI */
/* traditional - K&R */

void nrerror();
float *vector();
Rest of traditional declarations are here on the diskette.

#endif /* ANSI */

#endif /* _NR_UTILS_H_ */
```

And here is `nrutil.c`:

```
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(char error_text[])
/* Numerical Recipes standard error handler */
{
```

```
    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

float *vector(long nl, long nh)
/* allocate a float vector with subscript range v[nl..nh] */
{
    float *v;

    v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}

int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{
    int *v;

    v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl+NR_END;
}

unsigned char *cvector(long nl, long nh)
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
    unsigned char *v;

    v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned char)));
    if (!v) nrerror("allocation failure in cvector()");
    return v-nl+NR_END;
}

unsigned long *lvector(long nl, long nh)
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
    unsigned long *v;

    v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
    if (!v) nrerror("allocation failure in lvector()");
    return v-nl+NR_END;
}

double *dvector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
    double *v;

    v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl+NR_END;
}

float **matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    float **m;

    /* allocate pointers to rows */
    m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
```

```
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    /* allocate rows and set pointers to them */
    m[nrl]=(float *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

double **dmatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    double **m;

    /* allocate pointers to rows */
    m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    /* allocate rows and set pointers to them */
    m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

int **imatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    int **m;

    /* allocate pointers to rows */
    m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;


    /* allocate rows and set pointers to them */
    m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}
```

```
float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,
	long newrl, long newcl)
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
	long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
	float **m;

	/* allocate array of pointers to rows */
	m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
	if (!m) nrerror("allocation failure in submatrix()");
	m += NR_END;
	m -= newrl;

	/* set pointers to rows */
	for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}

float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
	long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
	float **m;

	/* allocate pointers to rows */
	m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
	if (!m)	nrerror("allocation failure in convert_matrix()");
	m += NR_END;
	m -= nrl;

	/* set pointers to rows */
	m[nrl]=a-ncl;
	for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
	/* return pointer to array of pointers to rows */
	return m;
}

float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh)
/* allocate a float 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */
{
	long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
	float ***t;

	/* allocate pointers to pointers to rows */
	t=(float ***) malloc((size_t)((nrow+NR_END)*sizeof(float**)));
	if (!t) nrerror("allocation failure 1 in f3tensor()");
	t += NR_END;
	t -= nrl;

	/* allocate pointers to rows and set pointers to them */
	t[nrl]=(float **) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float*)));
	if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");
	t[nrl] += NR_END;
	t[nrl] -= ncl;

	/* allocate rows and set pointers to them */
	t[nrl][ncl]=(float *) malloc((size_t)((nrow*ncol*ndep+NR_END)*sizeof(float)));
	if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
```

```
    t[nrl][ncl] += NR_END;
    t[nrl][ncl] -= ndl;

    for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
    for(i=nrl+1;i<=nrh;i++) {
        t[i]=t[i-1]+ncol;
        t[i][ncl]=t[i-1][ncl]+ncol*ndep;
        for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
    }

    /* return pointer to array of pointers to rows */
    return t;
}

void free_vector(float *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_ivector(int *v, long nl, long nh)
/* free an int vector allocated with ivector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_cvector(unsigned char *v, long nl, long nh)
/* free an unsigned char vector allocated with cvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_lvector(unsigned long *v, long nl, long nh)
/* free an unsigned long vector allocated with lvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_dvector(double *v, long nl, long nh)
/* free a double vector allocated with dvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
/* free a float matrix allocated by matrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)
/* free a double matrix allocated by dmatrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
/* free an int matrix allocated by imatrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}
```

```
void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a submatrix allocated by submatrix() */
{
    free((FREE_ARG) (b+nrl-NR_END));
}

void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a matrix allocated by convert_matrix() */
{
    free((FREE_ARG) (b+nrl-NR_END));
}

void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
    long ndl, long ndh)
/* free a float f3tensor allocated by f3tensor() */
{
    free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
    free((FREE_ARG) (t[nrl]+ncl-NR_END));
    free((FREE_ARG) (t+nrl-NR_END));
}
```

Appendix C: Complex Arithmetic

The functions listed below are used by the Recipes `cisi`, `frenel`, `hypdrv`, `hypgeo`, `hypser`, `laguer`, `zroots`, and `fixrts` for complex arithmetic. A complex number is defined to be a structure containing two `float` values, the real (`.r`) and imaginary (`.i`) parts. Complex arguments are passed and returned *by value*. See additional discussion in §1.2.

On the diskette, this Appendix is in the file `complex.c`.

```
#include <math.h>

typedef struct FCOMPLEX {float r,i;} fcomplex;

fcomplex Cadd(fcomplex a, fcomplex b)
{
    fcomplex c;
    c.r=a.r+b.r;
    c.i=a.i+b.i;
    return c;
}

fcomplex Csub(fcomplex a, fcomplex b)
{
    fcomplex c;
    c.r=a.r-b.r;
    c.i=a.i-b.i;
    return c;
}


fcomplex Cmul(fcomplex a, fcomplex b)
{
    fcomplex c;
    c.r=a.r*b.r-a.i*b.i;
    c.i=a.i*b.r+a.r*b.i;
    return c;
}

fcomplex Complex(float re, float im)
{
    fcomplex c;
    c.r=re;
    c.i=im;
    return c;
}

fcomplex Conjg(fcomplex z)
{
    fcomplex c;
    c.r=z.r;
```

```
    c.i = -z.i;
    return c;
}

fcomplex Cdiv(fcomplex a, fcomplex b)
{
    fcomplex c;
    float r,den;
    if (fabs(b.r) >= fabs(b.i)) {
        r=b.i/b.r;
        den=b.r+r*b.i;
        c.r=(a.r+r*a.i)/den;
        c.i=(a.i-r*a.r)/den;
    } else {
        r=b.r/b.i;
        den=b.i+r*b.r;
        c.r=(a.r*r+a.i)/den;
        c.i=(a.i*r-a.r)/den;
    }
    return c;
}

float Cabs(fcomplex z)
{
    float x,y,ans,temp;
    x=fabs(z.r);
    y=fabs(z.i);
    if (x == 0.0)
        ans=y;
    else if (y == 0.0)
        ans=x;
    else if (x > y) {
        temp=y/x;
        ans=x*sqrt(1.0+temp*temp);
    } else {
        temp=x/y;
        ans=y*sqrt(1.0+temp*temp);
    }
    return ans;
}

fcomplex Csqrt(fcomplex z)
{
    fcomplex c;
    float x,y,w,r;
    if ((z.r == 0.0) && (z.i == 0.0)) {
        c.r=0.0;
        c.i=0.0;
        return c;
    } else {
        x=fabs(z.r);
        y=fabs(z.i);
        if (x >= y) {
            r=y/x;
            w=sqrt(x)*sqrt(0.5*(1.0+sqrt(1.0+r*r)));
        } else {
            r=x/y;
            w=sqrt(y)*sqrt(0.5*(r+sqrt(1.0+r*r)));
        }
        if (z.r >= 0.0) {
            c.r=w;
            c.i=z.i/(2.0*w);
        } else {
            c.i=(z.i >= 0) ? w : -w;
```

```
            c.r=z.i/(2.0*c.i);
        }
        return c;
    }
}

fcomplex RCmul(float x, fcomplex a)
{
    fcomplex c;
    c.r=x*a.r;
    c.i=x*a.i;
    return c;
}
```

Index of Programs and Dependencies

The following table lists, in alphabetical order, all the routines in *Numerical Recipes*. When a routine requires subsidiary routines, either from this book or else user-supplied, the full dependency tree is shown: A routine calls directly all routines to which it is connected by a solid line in the column immediately to its right; it calls indirectly the connected routines in all columns to its right. Typographical conventions: Routines from this book are in typewriter font (e.g., `eulsum`, *gammln*). The smaller, slanted font is used for the second and subsequent occurences of a routine in a single dependency tree. (When you are getting routines from the *Numerical Recipes* diskettes, or their archive files, you need only specify names in the larger, upright font.) User-supplied routines are indicated by the use of text font and square brackets, e.g., [funcv]. Consult the text for individual specifications of these routines. The right-hand side of the table lists section and page numbers for each program.