

Appendix A: Table of Function Declarations

We here list the function declarations for all the routines in *Numerical Recipes in C++*. You should `#include` this listing in each separately compiled source file that contains or references any routine from this book. You have to apply the scope resolution operator `NR::` either explicitly or implicitly to access a routine from the `NR` namespace (see §1.2).

In the machine-readable distribution of *Numerical Recipes* (e.g., CDROM or Internet download), this Appendix is in the file `nr.h`. Note that it automatically includes the files `nrutil.h` and `nrtypes.h` that are detailed in Appendix B, making those facilities available to your program.

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

```
#ifndef _NR_H_
#define _NR_H_
#include <fstream>
#include <complex>
#include "nrutil.h"
#include "nrtypes.h"
using namespace std;

namespace NR {

void addint(Mat_O_DP &uf, Mat_I_DP &uc, Mat_O_DP &res);
void airy(const DP x, DP &ai, DP &bi, DP &aip, DP &bip);
void ambsa(Mat_IO_DP &p, Vec_IO_DP &y, Vec_O_DP &pb, DP &yb, const DP ftol,
    DP funk(Vec_I_DP &), int &iter, const DP temptr);
void amoeba(Mat_IO_DP &p, Vec_IO_DP &y, const DP ftol, DP funk(Vec_I_DP &),
    int &nfunk);
DP amotry(Mat_IO_DP &p, Vec_O_DP &y, Vec_IO_DP &psum, DP funk(Vec_I_DP &),
    const int ihi, const DP fac);
DP amotsa(Mat_IO_DP &p, Vec_O_DP &y, Vec_IO_DP &psum, Vec_O_DP &pb, DP &yb,
    DP funk(Vec_I_DP &), const int ihi, DP &yhi, const DP fac);
void anneal(Vec_I_DP &x, Vec_I_DP &y, Vec_IO_INT &iorder);
DP anorm2(Mat_I_DP &a);
void arcmak(Vec_I_ULNG &nfreq, unsigned long nchh, unsigned long nradd,
    arithcode &acode);
void arcode(unsigned long &ich, string &code, unsigned long &lcd,
    const int isign, arithcode &acode);
void arcsun(Vec_I_ULNG &iin, Vec_O_ULNG &iout, unsigned long ja,
    const int nwk, const unsigned long nrad, const unsigned long nc);
void asolve(Vec_I_DP &b, Vec_O_DP &x, const int itrns);
void atimes(Vec_I_DP &x, Vec_O_DP &r, const int itrns);
void avevar(Vec_I_DP &data, DP &ave, DP &var);
```

```

void balanc(Mat_IO_DP &a);
void banbks(Mat_I_DP &a, const int m1, const int m2, Mat_I_DP &al,
  Vec_I_INT &indx, Vec_IO_DP &b);
void bandec(Mat_IO_DP &a, const int m1, const int m2, Mat_O_DP &al,
  Vec_O_INT &indx, DP &d);
void banmul(Mat_I_DP &a, const int m1, const int m2, Vec_I_DP &x,
  Vec_O_DP &b);
void bcucof(Vec_I_DP &y, Vec_I_DP &y1, Vec_I_DP &y2, Vec_I_DP &y12,
  const DP d1, const DP d2, Mat_O_DP &c);
void bcuint(Vec_I_DP &y, Vec_I_DP &y1, Vec_I_DP &y2, Vec_I_DP &y12,
  const DP x1l, const DP x1u, const DP x2l, const DP x2u,
  const DP x1, const DP x2, DP &ansy, DP &ansy1, DP &ansy2);
void beschb(const DP x, DP &gam1, DP &gam2, DP &gampl, DP &gammi);
DP bessi(const int n, const DP x);
DP bessi0(const DP x);
DP bessi1(const DP x);
void bessik(const DP x, const DP xnu, DP &ri, DP &rk, DP &rip, DP &rkp);
DP bessj(const int n, const DP x);
DP bessj0(const DP x);
DP bessj1(const DP x);
void bessjy(const DP x, const DP xnu, DP &rj, DP &ry, DP &rjp, DP &ryp);
DP bessk(const int n, const DP x);
DP bessk0(const DP x);
DP bessk1(const DP x);
DP bessy(const int n, const DP x);
DP bessy0(const DP x);
DP bessy1(const DP x);
DP beta(const DP z, const DP w);
DP betacf(const DP a, const DP b, const DP x);
DP betai(const DP a, const DP b, const DP x);
DP bico(const int n, const int k);
void bksub(const int ne, const int nb, const int jf, const int k1,
  const int k2, Mat3D_IO_DP &c);
DP bnldev(const DP pp, const int n, int &idum);
DP brent(const DP ax, const DP bx, const DP cx, DP f(const DP),
  const DP tol, DP &xmin);
void broydn(Vec_IO_DP &x, bool &check, void vecfunc(Vec_I_DP &, Vec_O_DP &));
void bsstep(Vec_IO_DP &y, Vec_IO_DP &dydx, DP &xx, const DP htry,
  const DP eps, Vec_I_DP &yscal, DP &hdid, DP &hnext,
  void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void caldat(const int julian, int &mm, int &id, int &iyyy);
void chder(const DP a, const DP b, Vec_I_DP &c, Vec_O_DP &cder, const int n);
DP chebev(const DP a, const DP b, Vec_I_DP &c, const int m, const DP x);
void chebft(const DP a, const DP b, Vec_O_DP &c, DP func(const DP));
void chebpc(Vec_I_DP &c, Vec_O_DP &d);
void chint(const DP a, const DP b, Vec_I_DP &c, Vec_O_DP &cint, const int n);
DP chixy(const DP bang);
void choldc(Mat_IO_DP &a, Vec_O_DP &p);
void cholsl(Mat_I_DP &a, Vec_I_DP &p, Vec_I_DP &b, Vec_O_DP &x);
void chsone(Vec_I_DP &bins, Vec_I_DP &ebins, const int knstrn, DP &df,
  DP &chsqs, DP &prob);
void chstwo(Vec_I_DP &bins1, Vec_I_DP &bins2, const int knstrn, DP &df,
  DP &chsqs, DP &prob);
void cisi(const DP x, complex<DP> &cs);
void cntab1(Mat_I_INT &nn, DP &chisq, DP &df, DP &prob, DP &cramrv, DP &ccc);
void cntab2(Mat_I_INT &nn, DP &h, DP &hx, DP &hy, DP &hygx, DP &hxgy,
  DP &uygx, DP &uxgy, DP &uxy);
void convlv(Vec_I_DP &data, Vec_I_DP &respns, const int isign,
  Vec_O_DP &ans);
void copy(Mat_O_DP &aout, Mat_I_DP &ain);
void correl(Vec_I_DP &data1, Vec_I_DP &data2, Vec_O_DP &ans);
void cosft1(Vec_IO_DP &y);
void cosft2(Vec_IO_DP &y, const int isign);
void covsrt(Mat_IO_DP &covar, Vec_I_BOOL &ia, const int mfit);

```

```

void crank(Vec_IO_DP &w, DP &s);
void cyclic(Vec_I_DP &a, Vec_I_DP &b, Vec_I_DP &c, const DP alpha,
  const DP beta, Vec_I_DP &r, Vec_O_DP &x);
void daub4(Vec_IO_DP &a, const int n, const int isign);
DP dawson(const DP x);
DP dbrent(const DP ax, const DP bx, const DP cx, DP f(const DP),
  DP df(const DP), const DP tol, DP &xmin);
void ddpoly(Vec_I_DP &c, const DP x, Vec_O_DP &pd);
bool decchk(string str, char &ch);
void derivs_s(const DP x, Vec_I_DP &y, Vec_O_DP &dydx);
DP dfdim(const DP x);
void dfpmin(Vec_IO_DP &p, const DP gtol, int &iter, DP &fret,
  DP func(Vec_I_DP &), void dfunc(Vec_I_DP &, Vec_O_DP &));
DP dfridr(DP func(const DP), const DP x, const DP h, DP &err);
void dftcor(const DP w, const DP delta, const DP a, const DP b,
  Vec_I_DP &endpts, DP &corre, DP &corim, DP &corfac);
void dftint(DP func(const DP), const DP a, const DP b, const DP w,
  DP &cosint, DP &sinint);
void difeq(const int k, const int k1, const int k2, const int jsf,
  const int is1, const int isf, Vec_I_INT &indexv, Mat_O_DP &s,
  Mat_I_DP &y);
void dlinmin(Vec_IO_DP &p, Vec_IO_DP &xi, DP &fret, DP func(Vec_I_DP &),
  void dfunc(Vec_I_DP &, Vec_O_DP &));
void eclass(Vec_O_INT &nf, Vec_I_INT &lista, Vec_I_INT &listb);
void eclazz(Vec_O_INT &nf, bool equiv(const int, const int));
DP ei(const DP x);
void eigsrt(Vec_IO_DP &d, Mat_IO_DP &v);
DP elle(const DP phi, const DP ak);
DP ellf(const DP phi, const DP ak);
DP ellpi(const DP phi, const DP en, const DP ak);
void elmhes(Mat_IO_DP &a);
DP erfcc(const DP x);
DP erff(const DP x);
DP erffc(const DP x);
DP eulsum(DP &sum, const DP term, const int jterm, Vec_IO_DP &wksp);
DP evlmem(const DP fdt, Vec_I_DP &d, const DP xms);
DP expdev(int &idum);
DP expint(const int n, const DP x);
DP fdim(const DP x);
DP factln(const int n);
DP factrl(const int n);
void fasper(Vec_I_DP &x, Vec_I_DP &y, const DP ofac, const DP hifac,
  Vec_O_DP &wk1, Vec_O_DP &wk2, int &nout, int &jmax, DP &prob);
void fdjac(Vec_IO_DP &x, Vec_I_DP &fvec, Mat_O_DP &df,
  void vecfunc(Vec_I_DP &, Vec_O_DP &));
void fgauss(const DP x, Vec_I_DP &a, DP &y, Vec_O_DP &dyda);
void fit(Vec_I_DP &x, Vec_I_DP &y, Vec_I_DP &sig, const bool mwt, DP &a,
  DP &b, DP &sig_a, DP &sig_b, DP &chi2, DP &q);
void fitxy(Vec_I_DP &x, Vec_I_DP &y, Vec_I_DP &sigx, Vec_I_DP &sigy,
  DP &a, DP &b, DP &sig_a, DP &sig_b, DP &chi2, DP &q);
void fixrts(Vec_IO_DP &d);
void fleg(const DP x, Vec_O_DP &pl);
void flmoon(const int n, const int nph, int &jd, DP &frac);
DP fmin(Vec_I_DP &x);
void four1(Vec_IO_DP &data, const int isign);
void fourew(Vec_FSTREAM_p &file, int &na, int &nb, int &nc, int &nd);
void fourfs(Vec_FSTREAM_p &file, Vec_I_INT &n, const int isign);
void fourn(Vec_IO_DP &data, Vec_I_INT &n, const int isign);
void fpoly(const DP x, Vec_O_DP &p);
void fred2(const DP a, const DP b, Vec_O_DP &t, Vec_O_DP &f, Vec_O_DP &w,
  DP g(const DP), DP ak(const DP, const DP));
DP fredin(const DP x, const DP a, const DP b, Vec_I_DP &t, Vec_I_DP &f,
  Vec_I_DP &w, DP g(const DP), DP ak(const DP, const DP));
void frenel(const DP x, complex<DP> &cs);

```

```

void frprmn(Vec_IO_DP &p, const DP ftol, int &iter, DP &fret,
    DP func(Vec_I_DP &), void dfunc(Vec_I_DP &, Vec_O_DP &));
void ftest(Vec_I_DP &data1, Vec_I_DP &data2, DP &f, DP &prob);
DP gamdev(const int ia, int &idum);
DP gammln(const DP xx);
DP gammp(const DP a, const DP x);
DP gammq(const DP a, const DP x);
DP gasdev(int &idum);
void gaucof(Vec_IO_DP &a, Vec_IO_DP &b, const DP amu0, Vec_O_DP &x,
    Vec_O_DP &w);
void gauher(Vec_O_DP &x, Vec_O_DP &w);
void gaujac(Vec_O_DP &x, Vec_O_DP &w, const DP alf, const DP bet);
void gaulag(Vec_O_DP &x, Vec_O_DP &w, const DP alf);
void gauleg(const DP x1, const DP x2, Vec_O_DP &x, Vec_O_DP &w);
void gaussj(Mat_IO_DP &a, Mat_IO_DP &b);
void gcf(DP &gamcfc, const DP a, const DP x, DP &gln);
DP golden(const DP ax, const DP bx, const DP cx, DP f(const DP),
    const DP tol, DP &xmin);
void gser(DP &gamser, const DP a, const DP x, DP &gln);
void hpsel(Vec_I_DP &arr, Vec_O_DP &heap);
void hpsort(Vec_IO_DP &ra);
void hqr(Mat_IO_DP &a, Vec_O_CPLX_DP &wri);
void hufapp(Vec_IO_ULNG &index, Vec_I_ULNG &nprob, const unsigned long n,
    const unsigned long m);
void hufdec(unsigned long &ich, string &code, const unsigned long lcode,
    unsigned long &nb, huffcode &hcode);
void hufenc(const unsigned long ich, string &code, unsigned long &nb,
    huffcode &hcode);
void hufmak(Vec_I_ULNG &nfreq, const unsigned long nchin,
    unsigned long &ilong, unsigned long &nlong, huffcode &hcode);
void hunt(Vec_I_DP &xx, const DP x, int &jlo);
void hypdrv(const DP s, Vec_I_DP &yy, Vec_O_DP &dyyds);
complex<DP> hypgeo(const complex<DP> &a, const complex<DP> &b,
    const complex<DP> &c, const complex<DP> &z);
void hypser(const complex<DP> &a, const complex<DP> &b,
    const complex<DP> &c, const complex<DP> &z,
    complex<DP> &series, complex<DP> &deriv);
unsigned short icrc(const unsigned short crc, const string &bufptr,
    const short jinit, const int jrev);
unsigned short icrc1(const unsigned short crc, const unsigned char onech);
unsigned long igray(const unsigned long n, const int is);
void indexx(Vec_I_DP &arr, Vec_O_INT &indx);
void indexx(Vec_I_INT &arr, Vec_O_INT &indx);
void interp(Mat_O_DP &uf, Mat_I_DP &uc);
int irbit1(unsigned long &iseed);
int irbit2(unsigned long &iseed);
void jacobi(Mat_IO_DP &a, Vec_O_DP &d, Mat_O_DP &v, int &nrot);
void jacobn_s(const DP x, Vec_I_DP &y, Vec_O_DP &dfdx, Mat_O_DP &dfdy);
int julday(const int mm, const int id, const int iyyy);
void kend11(Vec_I_DP &data1, Vec_I_DP &data2, DP &tau, DP &z, DP &prob);
void kend12(Mat_I_DP &tab, DP &tau, DP &z, DP &prob);
void kermom(Vec_O_DP &w, const DP y);
void ks2d1s(Vec_I_DP &x1, Vec_I_DP &y1, void quadvl(const DP, const DP,
    DP &, DP &, DP &, DP &), DP &d1, DP &prob);
void ks2d2s(Vec_I_DP &x1, Vec_I_DP &y1, Vec_I_DP &x2, Vec_I_DP &y2, DP &d,
    DP &prob);
void ksone(Vec_IO_DP &data, DP func(const DP), DP &d, DP &prob);
void kstwo(Vec_IO_DP &data1, Vec_IO_DP &data2, DP &d, DP &prob);
void laguer(Vec_I_CPLX_DP &a, complex<DP> &x, int &its);
void lfit(Vec_I_DP &x, Vec_I_DP &y, Vec_I_DP &sig, Vec_IO_DP &a,
    Vec_I_BOOL &ia, Mat_O_DP &covar, DP &chisq,
    void funcs(const DP, Vec_O_DP &));
void linbcg(Vec_I_DP &b, Vec_IO_DP &x, const int itol, const DP tol,
    const int itmax, int &iter, DP &err);

```

```

void linmin(Vec_IO_DP &p, Vec_IO_DP &xi, DP &fret, DP func(Vec_I_DP &));
void lnsrch(Vec_I_DP &xold, const DP fold, Vec_I_DP &g, Vec_IO_DP &p,
  Vec_O_DP &x, DP &f, const DP stpmax, bool &check, DP func(Vec_I_DP &));
void locate(Vec_I_DP &xxx, const DP x, int &j);
void lop(Mat_O_DP &out, Mat_I_DP &u);
void lubksb(Mat_I_DP &a, Vec_I_INT &indx, Vec_IO_DP &b);
void ludcmp(Mat_IO_DP &a, Vec_O_INT &indx, DP &d);
void machar(int &ibeta, int &it, int &irnd, int &ngrd, int &machep,
  int &negep, int &ieexp, int &minexp, int &maxexp, DP &eps, DP &epsneg,
  DP &xmin, DP &xmax);
void matadd(Mat_I_DP &a, Mat_I_DP &b, Mat_O_DP &c);
void matsub(Mat_I_DP &a, Mat_I_DP &b, Mat_O_DP &c);
void medfit(Vec_I_DP &x, Vec_I_DP &y, DP &a, DP &b, DP &abdev);
void memcof(Vec_I_DP &data, DP &xms, Vec_O_DP &d);
bool metrop(const DP de, const DP t);
void mgfas(Mat_IO_DP &u, const int maxcyc);
void mglin(Mat_IO_DP &u, const int ncycle);
DP midexp(DP funk(const DP), const DP aa, const DP bb, const int n);
DP midinf(DP funk(const DP), const DP aa, const DP bb, const int n);
DP midpnt(DP funk(const DP), const DP a, const DP b, const int n);
DP midsql(DP funk(const DP), const DP aa, const DP bb, const int n);
DP midsqu(DP funk(const DP), const DP aa, const DP bb, const int n);
void miser(DP func(Vec_I_DP &), Vec_I_DP &regn, const int npts,
  const DP dith, DP &ave, DP &var);
void mmid(Vec_I_DP &y, Vec_I_DP &dydx, const DP xs, const DP htot,
  const int nstep, Vec_O_DP &yout,
  void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void mnbrak(DP &ax, DP &bx, DP &cx, DP &fa, DP &fb, DP &fc,
  DP func(const DP));
void mnewt(const int ntrial, Vec_IO_DP &x, const DP tolx, const DP tolf);
void moment(Vec_I_DP &data, DP &ave, DP &adev, DP &sdev, DP &var, DP &skew,
  DP &curt);
void mp2dfr(Vec_IO_UCHR &a, string &s);
void mpadd(Vec_O_UCHR &w, Vec_I_UCHR &u, Vec_I_UCHR &v);
void mpdiv(Vec_O_UCHR &q, Vec_O_UCHR &r, Vec_I_UCHR &u, Vec_I_UCHR &v);
void mpinv(Vec_O_UCHR &u, Vec_I_UCHR &v);
void mplsh(Vec_IO_UCHR &u);
void mpmov(Vec_O_UCHR &u, Vec_I_UCHR &v);
void mpmul(Vec_O_UCHR &w, Vec_I_UCHR &u, Vec_I_UCHR &v);
void mpneg(Vec_IO_UCHR &u);
void mppi(const int np);
void mprove(Mat_I_DP &a, Mat_I_DP &alud, Vec_I_INT &indx, Vec_I_DP &b,
  Vec_IO_DP &x);
void mpsad(Vec_O_UCHR &w, Vec_I_UCHR &u, const int iv);
void mpsdv(Vec_O_UCHR &w, Vec_I_UCHR &u, const int iv, int &ir);
void mpsmu(Vec_O_UCHR &w, Vec_I_UCHR &u, const int iv);
void mpsqrt(Vec_O_UCHR &w, Vec_O_UCHR &u, Vec_I_UCHR &v);
void mpsub(int &is, Vec_O_UCHR &w, Vec_I_UCHR &u, Vec_I_UCHR &v);
void mrqcof(Vec_I_DP &x, Vec_I_DP &y, Vec_I_DP &sig, Vec_I_DP &a,
  Vec_I_BOOL &ia, Mat_O_DP &alpha, Vec_O_DP &beta, DP &chisq,
  void funcs(const DP, Vec_I_DP &, DP &, Vec_O_DP &));
void mrqmin(Vec_I_DP &x, Vec_I_DP &y, Vec_I_DP &sig, Vec_IO_DP &a,
  Vec_I_BOOL &ia, Mat_O_DP &covar, Mat_O_DP &alpha, DP &chisq,
  void funcs(const DP, Vec_I_DP &, DP &, Vec_O_DP &), DP &alamda);
void newt(Vec_IO_DP &x, bool &check, void vecfunc(Vec_I_DP &, Vec_O_DP &));
void odeint(Vec_IO_DP &ystart, const DP x1, const DP x2, const DP eps,
  const DP h1, const DP hmin, int &nok, int &nbad,
  void derivs(const DP, Vec_I_DP &, Vec_O_DP &),
  void rkqs(Vec_IO_DP &, Vec_IO_DP &, DP &, const DP, const DP,
  Vec_I_DP &, DP &, DP &, void (*)(const DP, Vec_I_DP &, Vec_O_DP &));
void orthog(Vec_I_DP &anu, Vec_I_DP &alpha, Vec_I_DP &beta, Vec_O_DP &a,
  Vec_O_DP &b);
void pade(Vec_IO_DP &cof, DP &resid);
void pccheb(Vec_I_DP &d, Vec_O_DP &c);

```

```

void pcshtft(const DP a, const DP b, Vec_IO_DP &d);
void pearsn(Vec_I_DP &x, Vec_I_DP &y, DP &r, DP &prob, DP &z);
void period(Vec_I_DP &x, Vec_I_DP &y, const DP ofac, const DP hifac,
  Vec_O_DP &px, Vec_O_DP &py, int &nout, int &jmax, DP &prob);
void piksr2(Vec_IO_DP &arr, Vec_IO_DP &brr);
void piksrt(Vec_IO_DP &arr);
void pinvs(const int ie1, const int ie2, const int je1, const int jsf,
  const int jcl, const int k, Mat3D_O_DP &c, Mat_IO_DP &s);
DP plgnr(const int l, const int m, const DP x);
DP poidev(const DP xm, int &idum);
void polcoe(Vec_I_DP &x, Vec_I_DP &y, Vec_O_DP &cof);
void polcof(Vec_I_DP &xa, Vec_I_DP &ya, Vec_O_DP &cof);
void poldiv(Vec_I_DP &u, Vec_I_DP &v, Vec_O_DP &q, Vec_O_DP &r);
void polin2(Vec_I_DP &x1a, Vec_I_DP &x2a, Mat_I_DP &ya, const DP x1,
  const DP x2, DP &y, DP &dy);
void polint(Vec_I_DP &xa, Vec_I_DP &ya, const DP x, DP &y, DP &dy);
void powell(Vec_IO_DP &p, Mat_IO_DP &xi, const DP ftol, int &iter,
  DP &fret, DP func(Vec_I_DP &));
void predic(Vec_I_DP &data, Vec_I_DP &d, Vec_O_DP &future);
DP probks(const DP alam);
void psdes(unsigned long &lword, unsigned long &irword);
void pwt(Vec_IO_DP &a, const int n, const int isign);
void pwtset(const int n);
DP pythag(const DP a, const DP b);
void pzextr(const int iest, const DP xest, Vec_I_DP &yst, Vec_O_DP &yz,
  Vec_O_DP &dy);
DP qgaus(DP func(const DP), const DP a, const DP b);
void qrdcmp(Mat_IO_DP &a, Vec_O_DP &c, Vec_O_DP &d, bool &sing);
DP qromb(DP func(const DP), DP a, DP b);
DP qromo(DP func(const DP), const DP a, const DP b,
  DP choose(DP (*)(const DP), const DP, const DP, const int));
void qroot(Vec_I_DP &p, DP &b, DP &c, const DP eps);
void qrslv(Mat_I_DP &a, Vec_I_DP &c, Vec_I_DP &d, Vec_IO_DP &b);
void qrupdt(Mat_IO_DP &r, Mat_IO_DP &qt, Vec_IO_DP &u, Vec_I_DP &v);
DP qsimp(DP func(const DP), const DP a, const DP b);
DP qtrap(DP func(const DP), const DP a, const DP b);
DP quad3d(DP func(const DP, const DP, const DP), const DP x1, const DP x2);
void quadct(const DP x, const DP y, Vec_I_DP &xx, Vec_I_DP &yy, DP &fa,
  DP &fb, DP &fc, DP &fd);
void quadmx(Mat_O_DP &a);
void quadvl(const DP x, const DP y, DP &fa, DP &fb, DP &fc, DP &fd);
DP ran0(int &idum);
DP ran1(int &idum);
DP ran2(int &idum);
DP ran3(int &idum);
DP ran4(int &idum);
void rank(Vec_I_INT &indx, Vec_O_INT &irank);
void ranpt(Vec_O_DP &pt, Vec_I_DP &regn);
void ratint(Vec_I_DP &xa, Vec_I_DP &ya, const DP x, DP &y, DP &dy);
void ratlsq(DP fn(const DP), const DP a, const DP b, const int mm,
  const int kk, Vec_O_DP &cof, DP &dev);
DP ratval(const DP x, Vec_I_DP &cof, const int mm, const int kk);
DP rc(const DP x, const DP y);
DP rd(const DP x, const DP y, const DP z);
void realft(Vec_IO_DP &data, const int isign);
void rebin(const DP rc, const int nd, Vec_I_DP &r, Vec_O_DP &xin,
  Mat_IO_DP &xi, const int j);
void red(const int iz1, const int iz2, const int jz1, const int jz2,
  const int jm1, const int jm2, const int jmf, const int ic1,
  const int jcl, const int jcf, const int kc, Mat3D_I_DP &c,
  Mat_IO_DP &s);
void relax(Mat_IO_DP &u, Mat_I_DP &rhs);
void relax2(Mat_IO_DP &u, Mat_I_DP &rhs);
void resid(Mat_O_DP &res, Mat_I_DP &u, Mat_I_DP &rhs);

```

```

DP revcst(Vec_I_DP &x, Vec_I_DP &y, Vec_I_INT &iorder, Vec_IO_INT &n);
void reverse(Vec_IO_INT &iorder, Vec_I_INT &n);
DP rf(const DP x, const DP y, const DP z);
DP rj(const DP x, const DP y, const DP z, const DP p);
void rk4(Vec_I_DP &y, Vec_I_DP &dydx, const DP x, const DP h,
        Vec_O_DP &yout, void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void rkck(Vec_I_DP &y, Vec_I_DP &dydx, const DP x,
        const DP h, Vec_O_DP &yout, Vec_O_DP &yerr,
        void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void rkdump(Vec_I_DP &vstart, const DP x1, const DP x2,
        void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void rkqs(Vec_IO_DP &y, Vec_IO_DP &dydx, DP &x, const DP htry,
        const DP eps, Vec_I_DP &yscal, DP &hdid, DP &hnext,
        void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void rlft3(Mat3D_IO_DP &data, Mat_IO_DP &speq, const int isign);
DP rofunc(const DP b);
void rotate(Mat_IO_DP &r, Mat_IO_DP &qt, const int i, const DP a,
        const DP b);
void rsolv(Mat_I_DP &a, Vec_I_DP &d, Vec_IO_DP &b);
void rstrct(Mat_O_DP &uc, Mat_I_DP &uf);
DP rtbis(DP func(const DP), const DP x1, const DP x2, const DP xacc);
DP rtfisp(DP func(const DP), const DP x1, const DP x2, const DP xacc);
DP rtnewt(void funcd(const DP, DP &, DP &), const DP x1, const DP x2,
        const DP xacc);
DP rtsafe(void funcd(const DP, DP &, DP &), const DP x1, const DP x2,
        const DP xacc);
DP rtsec(DP func(const DP), const DP x1, const DP x2, const DP xacc);
void rzextr(const int iest, const DP xest, Vec_I_DP &yest, Vec_O_DP &yz,
        Vec_O_DP &dy);
void savgol(Vec_O_DP &c, const int np, const int nl, const int nr,
        const int ld, const int m);
void scrsho(DP fx(const DP));
DP select(const int k, Vec_IO_DP &arr);
DP selip(const int k, Vec_I_DP &arr);
void shell(const int n, Vec_IO_DP &a);
void shoot(Vec_I_DP &v, Vec_O_DP &f);
void shootf(Vec_I_DP &v, Vec_O_DP &f);
void simp1(Mat_I_DP &a, const int mm, Vec_I_INT &ll, const int nll,
        const int iabf, int &kp, DP &bmax);
void simp2(Mat_I_DP &a, const int m, const int n, int &ip, const int kp);
void simp3(Mat_IO_DP &a, const int il, const int k1, const int ip,
        const int kp);
void simplx(Mat_IO_DP &a, const int m1, const int m2, const int m3,
        int &icase, Vec_O_INT &izrov, Vec_O_INT &iposv);
void simpr(Vec_I_DP &y, Vec_I_DP &dydx, Vec_I_DP &dwdx, Mat_I_DP &dwdy,
        const DP xs, const DP htot, const int nstep, Vec_O_DP &yout,
        void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void sinft(Vec_IO_DP &y);
void slvsm2(Mat_O_DP &u, Mat_I_DP &rhs);
void slvsm1(Mat_O_DP &u, Mat_I_DP &rhs);
void sncndn(const DP uu, const DP emmc, DP &sn, DP &cn, DP &dn);
DP snrm(Vec_I_DP &sx, const int itol);
void sobseq(const int n, Vec_O_DP &x);
void solvde(const int itmax, const DP conv, const DP slowc,
        Vec_I_DP &scalv, Vec_I_INT &indexv, const int nb, Mat_IO_DP &y);
void sor(Mat_I_DP &a, Mat_I_DP &b, Mat_I_DP &c, Mat_I_DP &d, Mat_I_DP &e,
        Mat_I_DP &f, Mat_IO_DP &u, const DP rjac);
void sort(Vec_IO_DP &arr);
void sort2(Vec_IO_DP &arr, Vec_IO_DP &brr);
void sort3(Vec_IO_DP &ara, Vec_IO_DP &rb, Vec_IO_DP &rc);
void spctrm(ifstream &fp, Vec_O_DP &p, const int k, const bool overlap);
void spear(Vec_I_DP &data1, Vec_I_DP &data2, DP &d, DP &zdz, DP &probd,
        DP &rs, DP &probrs);
void sphbes(const int n, const DP x, DP &sj, DP &sy, DP &sjp, DP &syp);

```

```

void splie2(Vec_I_DP &x1a, Vec_I_DP &x2a, Mat_I_DP &ya, Mat_O_DP &y2a);
void splin2(Vec_I_DP &x1a, Vec_I_DP &x2a, Mat_I_DP &ya, Mat_I_DP &y2a,
  const DP x1, const DP x2, DP &y);
void spline(Vec_I_DP &x, Vec_I_DP &y, const DP yp1, const DP ypn,
  Vec_O_DP &y2);
void splint(Vec_I_DP &xa, Vec_I_DP &ya, Vec_I_DP &y2a, const DP x, DP &y);
void spread(const DP y, Vec_IO_DP &yy, const DP x, const int m);
void sprsax(Vec_I_DP &sa, Vec_I_INT &ija, Vec_I_DP &x, Vec_O_DP &b);
void sprsin(Mat_I_DP &a, const DP thresh, Vec_O_DP &sa, Vec_O_INT &ija);
void sprspm(Vec_I_DP &sa, Vec_I_INT &ija, Vec_I_DP &sb, Vec_I_INT &ijb,
  Vec_O_DP &sc, Vec_I_INT &ijc);
void sprstm(Vec_I_DP &sa, Vec_I_INT &ija, Vec_I_DP &sb, Vec_I_INT &ijb,
  const DP thresh, Vec_O_DP &sc, Vec_O_INT &ijc);
void sprstp(Vec_I_DP &sa, Vec_I_INT &ija, Vec_O_DP &sb, Vec_O_INT &ijb);
void sprstx(Vec_I_DP &sa, Vec_I_INT &ija, Vec_I_DP &x, Vec_O_DP &b);
void stifbs(Vec_IO_DP &y, Vec_IO_DP &dydx, DP &xxx, const DP htry,
  const DP eps, Vec_I_DP &yscal, DP &hdid, DP &hnext,
  void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void stiff(Vec_IO_DP &y, Vec_IO_DP &dydx, DP &x, const DP htry,
  const DP eps, Vec_I_DP &yscal, DP &hdid, DP &hnext,
  void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void stoerm(Vec_I_DP &y, Vec_I_DP &d2y, const DP xs,
  const DP htot, const int nstep, Vec_O_DP &yout,
  void derivs(const DP, Vec_I_DP &, Vec_O_DP &));
void svbksb(Mat_I_DP &u, Vec_I_DP &w, Mat_I_DP &v, Vec_I_DP &b, Vec_O_DP &x);
void svdcmp(Mat_IO_DP &a, Vec_O_DP &w, Mat_O_DP &v);
void svdfit(Vec_I_DP &x, Vec_I_DP &y, Vec_I_DP &sig, Vec_O_DP &a,
  Mat_O_DP &u, Mat_O_DP &v, Vec_O_DP &w, DP &chisq,
  void funcs(const DP, Vec_O_DP &));
void svdvar(Mat_I_DP &v, Vec_I_DP &w, Mat_O_DP &cvm);
void toeplz(Vec_I_DP &r, Vec_O_DP &x, Vec_I_DP &y);
void tptest(Vec_I_DP &data1, Vec_I_DP &data2, DP &t, DP &prob);
void tqli(Vec_IO_DP &d, Vec_IO_DP &e, Mat_IO_DP &z);
DP trapzd(DP func(const DP), const DP a, const DP b, const int n);
void tred2(Mat_IO_DP &a, Vec_O_DP &d, Vec_O_DP &e);
void tridag(Vec_I_DP &a, Vec_I_DP &b, Vec_I_DP &c, Vec_I_DP &r, Vec_O_DP &u);
DP trncst(Vec_I_DP &x, Vec_I_DP &y, Vec_I_INT &iorder, Vec_IO_INT &n);
void trnspt(Vec_IO_INT &iorder, Vec_I_INT &n);
void ttest(Vec_I_DP &data1, Vec_I_DP &data2, DP &t, DP &prob);
void tutest(Vec_I_DP &data1, Vec_I_DP &data2, DP &t, DP &prob);
void twofit(Vec_I_DP &data1, Vec_I_DP &data2, Vec_O_DP &fft1,
  Vec_O_DP &fft2);
void vander(Vec_I_DP &x, Vec_O_DP &w, Vec_I_DP &q);
void vegas(Vec_I_DP &reg, DP fxn(Vec_I_DP &), const int init,
  const int ncall, const int itmx, const int nprn, DP &tgral, DP &sd,
  DP &chi2a);
void voltra(const DP t0, const DP h, Vec_O_DP &t, Mat_O_DP &f,
  DP g(const int, const DP),
  DP ak(const int, const int, const DP, const DP));
void wt1(Vec_IO_DP &a, const int isign,
  void wtstep(Vec_IO_DP &, const int, const int));
void wtn(Vec_IO_DP &a, Vec_I_INT &nn, const int isign,
  void wtstep(Vec_IO_DP &, const int, const int));
void wghts(Vec_O_DP &wghts, const DP h,
  void kermom(Vec_O_DP &w, const DP y));
bool zbrac(DP func(const DP), DP &x1, DP &x2);
void zbrak(DP fx(const DP), const DP x1, const DP x2, const int n,
  Vec_O_DP &xb1, Vec_O_DP &xb2, int &nroot);
DP zbrent(DP func(const DP), const DP x1, const DP x2, const DP tol);
void zrhqr(Vec_I_DP &a, Vec_O_CPLX_DP &rt);
DP zriddr(DP func(const DP), const DP x1, const DP x2, const DP xacc);
void zroots(Vec_I_CPLX_DP &a, Vec_O_CPLX_DP &roots, const bool &polish);
}
#endif /* _NR_H_ */

```

Appendix B: Utility Routines and Classes

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 and classes listed below are used by essentially all of the Recipes in this book. First we give the listing of the default set of typedefs (see §1.2). In the machine-readable distribution of *Numerical Recipes*, this is the file `nrtypes_nr.h`. In practice, however we include these definitions with the statement

```
#include "nrtypes.h"
```

The file `nrtypes.h` is an intermediate file that, itself, contains a single include statement, calling into play an “nrtypes” file that is specific to the matrix/vector class library you use. The default content of `nrtypes.h` is

```
#include "nrtypes_nr.h"
```

which has the effect of including the following file:

```
#ifndef _NR_TYPES_H_
#define _NR_TYPES_H_

#include <complex>
#include <fstream>
#include "nrutil.h"
using namespace std;

typedef double DP;

// Vector Types

typedef const NRVec<bool> Vec_I_BOOL;
typedef NRVec<bool> Vec_BOOL, Vec_O_BOOL, Vec_IO_BOOL;

typedef const NRVec<char> Vec_I_CHR;
typedef NRVec<char> Vec_CHR, Vec_O_CHR, Vec_IO_CHR;

typedef const NRVec<unsigned char> Vec_I_UCHR;
typedef NRVec<unsigned char> Vec_UCHR, Vec_O_UCHR, Vec_IO_UCHR;

typedef const NRVec<int> Vec_I_INT;
```

```

typedef NRVec<int> Vec_INT, Vec_O_INT, Vec_IO_INT;

typedef const NRVec<unsigned int> Vec_I_UINT;
typedef NRVec<unsigned int> Vec_UINT, Vec_O_UINT, Vec_IO_UINT;

typedef const NRVec<long> Vec_I_LNG;
typedef NRVec<long> Vec_LNG, Vec_O_LNG, Vec_IO_LNG;

typedef const NRVec<unsigned long> Vec_I_ULNG;
typedef NRVec<unsigned long> Vec_ULNG, Vec_O_ULNG, Vec_IO_ULNG;

typedef const NRVec<float> Vec_I_SP;
typedef NRVec<float> Vec_SP, Vec_O_SP, Vec_IO_SP;

typedef const NRVec<DP> Vec_I_DP;
typedef NRVec<DP> Vec_DP, Vec_O_DP, Vec_IO_DP;

typedef const NRVec<complex<float> > Vec_I_CPLX_SP;
typedef NRVec<complex<float> > Vec_CPLX_SP, Vec_O_CPLX_SP, Vec_IO_CPLX_SP;

typedef const NRVec<complex<DP> > Vec_I_CPLX_DP;
typedef NRVec<complex<DP> > Vec_CPLX_DP, Vec_O_CPLX_DP, Vec_IO_CPLX_DP;

// Matrix Types

typedef const NRMAT<bool> Mat_I_BOOL;
typedef NRMAT<bool> Mat_BOOL, Mat_O_BOOL, Mat_IO_BOOL;

typedef const NRMAT<char> Mat_I_CHR;
typedef NRMAT<char> Mat_CHR, Mat_O_CHR, Mat_IO_CHR;

typedef const NRMAT<unsigned char> Mat_I_UCHR;
typedef NRMAT<unsigned char> Mat_UCHR, Mat_O_UCHR, Mat_IO_UCHR;

typedef const NRMAT<int> Mat_I_INT;
typedef NRMAT<int> Mat_INT, Mat_O_INT, Mat_IO_INT;

typedef const NRMAT<unsigned int> Mat_I_UINT;
typedef NRMAT<unsigned int> Mat_UINT, Mat_O_UINT, Mat_IO_UINT;

typedef const NRMAT<long> Mat_I_LNG;
typedef NRMAT<long> Mat_LNG, Mat_O_LNG, Mat_IO_LNG;

typedef const NRVec<unsigned long> Mat_I_ULNG;
typedef NRMAT<unsigned long> Mat_ULNG, Mat_O_ULNG, Mat_IO_ULNG;

typedef const NRMAT<float> Mat_I_SP;
typedef NRMAT<float> Mat_SP, Mat_O_SP, Mat_IO_SP;

typedef const NRMAT<DP> Mat_I_DP;
typedef NRMAT<DP> Mat_DP, Mat_O_DP, Mat_IO_DP;

typedef const NRMAT<complex<float> > Mat_I_CPLX_SP;
typedef NRMAT<complex<float> > Mat_CPLX_SP, Mat_O_CPLX_SP, Mat_IO_CPLX_SP;

typedef const NRMAT<complex<DP> > Mat_I_CPLX_DP;
typedef NRMAT<complex<DP> > Mat_CPLX_DP, Mat_O_CPLX_DP, Mat_IO_CPLX_DP;

// 3D Matrix Types

typedef const NRMAT3d<DP> Mat3D_I_DP;
typedef NRMAT3d<DP> Mat3D_DP, Mat3D_O_DP, Mat3D_IO_DP;

// Miscellaneous Types

```

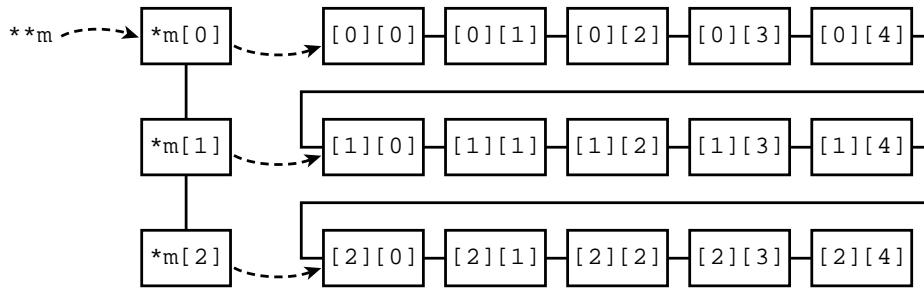


Figure B.1. Storage scheme for a matrix *m*, implemented as a pointer to an array of pointers to rows. Dotted lines denote address reference, while solid lines connect sequential memory locations.

```
typedef NRVec<unsigned long *> Vec_ULNG_p;
typedef NRVec<NRMat<DP> *> Vec_Mat_DP_p;
typedef NRVec<fstream *> Vec_FSTREAM_p;

#endif /* _NR_TYPES_H */
```

Next is a listing of the default vector and matrix classes, `NRVec` and `NRMat`, including the implementation (see §1.3).

Note that we do not use C-style arrays as the private data variables for vectors or matrices. For vectors we use a pointer to a block of storage that we explicitly allocate with `new`. For two-dimensional arrays, we use a pointer to an array of pointers, with the array elements pointing to the first element in the rows of the matrix (see Figure B.1). This scheme is much more flexible than fixed-size built-in arrays.

Also included in the listing are the utility routines `SQR`, `MAX`, `MIN`, `SIGN`, and `SWAP`, and the error routine `nrerror`, which is invoked to terminate program execution — with an appropriate message — when a fatal error is encountered.

In the machine-readable distribution of *Numerical Recipes*, this is the file `nrutil_nr.h`. In practice, however we include these definitions with the statement

```
#include "nrutil.h"
```

Like `nrtypes.h`, the file `nrutil.h` is an intermediate file that, itself, contains a single include statement, calling into play an “`nrutil`” file that is specific to the matrix/vector class library you use. The default content of `nrutil.h` is

```
#include "nrutil_nr.h"
```

which has the effect of including the following file:

```
#ifndef _NR_UTIL_H_
#define _NR_UTIL_H_

#include <string>
#include <cmath>
#include <complex>
#include <iostream>
using namespace std;

typedef double DP;
```

```

template<class T>
inline const T SQR(const T a) {return a*a;}

template<class T>
inline const T MAX(const T &a, const T &b)
{return b > a ? (b) : (a);}

inline float MAX(const double &a, const float &b)
{return b > a ? (b) : float(a);}

inline float MAX(const float &a, const double &b)
{return b > a ? float(b) : (a);}

template<class T>
inline const T MIN(const T &a, const T &b)
{return b < a ? (b) : (a);}

inline float MIN(const double &a, const float &b)
{return b < a ? (b) : float(a);}

inline float MIN(const float &a, const double &b)
{return b < a ? float(b) : (a);}

template<class T>
inline const T SIGN(const T &a, const T &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

inline float SIGN(const float &a, const double &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

inline float SIGN(const double &a, const float &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

template<class T>
inline void SWAP(T &a, T &b)
{T dum=a; a=b; b=dum;}

namespace NR {
  inline void nrerror(const string error_text)
  // Numerical Recipes standard error handler
  {
    cerr << "Numerical Recipes run-time error..." << endl;
    cerr << error_text << endl;
    cerr << "...now exiting to system..." << endl;
    exit(1);
  }
}

template <class T>
class NRVec {
private:
  int nn;    // size of array. upper index is nn-1
  T *v;
public:
  NRVec();
  explicit NRVec(int n);    // Zero-based array
  NRVec(const T &a, int n); //initialize to constant value
  NRVec(const T *a, int n); // Initialize to array
  NRVec(const NRVec &rhs);  // Copy constructor
  NRVec & operator=(const NRVec &rhs); //assignment
  NRVec & operator=(const T &a);    //assign a to every element
  inline T & operator[](const int i); //i'th element
  inline const T & operator[](const int i) const;
  inline int size() const;

```

```

    ~NRVec();
};

template <class T>
NRVec<T>::NRVec() : nn(0), v(0) {}

template <class T>
NRVec<T>::NRVec(int n) : nn(n), v(new T[n]) {}

template <class T>
NRVec<T>::NRVec(const T& a, int n) : nn(n), v(new T[n])
{
    for(int i=0; i<n; i++)
        v[i] = a;
}

template <class T>
NRVec<T>::NRVec(const T *a, int n) : nn(n), v(new T[n])
{
    for(int i=0; i<n; i++)
        v[i] = *a++;
}

template <class T>
NRVec<T>::NRVec(const NRVec<T> &rhs) : nn(rhs.nn), v(new T[nn])
{
    for(int i=0; i<nn; i++)
        v[i] = rhs[i];
}

template <class T>
NRVec<T> & NRVec<T>::operator=(const NRVec<T> &rhs)
// postcondition: normal assignment via copying has been performed;
//               if vector and rhs were different sizes, vector
//               has been resized to match the size of rhs
{
    if (this != &rhs)
    {
        if (nn != rhs.nn) {
            if (v != 0) delete [] (v);
            nn=rhs.nn;
            v= new T[nn];
        }
        for (int i=0; i<nn; i++)
            v[i]=rhs[i];
    }
    return *this;
}

template <class T>
NRVec<T> & NRVec<T>::operator=(const T &a) //assign a to every element
{
    for (int i=0; i<nn; i++)
        v[i]=a;
    return *this;
}

template <class T>
inline T & NRVec<T>::operator[](const int i) //subscripting
{
    return v[i];
}

template <class T>

```

```

inline const T & NRVec<T>::operator[](const int i) const //subscripting
{
    return v[i];
}

template <class T>
inline int NRVec<T>::size() const
{
    return nn;
}

template <class T>
NRVec<T>::~NRVec()
{
    if (v != 0)
        delete[] (v);
}

template <class T>
class NRMat {
private:
    int nn;
    int mm;
    T **v;
public:
    NRMat();
    NRMat(int n, int m); // Zero-based array
    NRMat(const T &a, int n, int m); //Initialize to constant
    NRMat(const T *a, int n, int m); // Initialize to array
    NRMat(const NRMat &rhs); // Copy constructor
    NRMat & operator=(const NRMat &rhs); //assignment
    NRMat & operator=(const T &a); //assign a to every element
    inline T* operator[](const int i); //subscripting: pointer to row i
    inline const T* operator[](const int i) const;
    inline int nrows() const;
    inline int ncols() const;
    ~NRMat();
};

template <class T>
NRMat<T>::NRMat() : nn(0), mm(0), v(0) {}

template <class T>
NRMat<T>::NRMat(int n, int m) : nn(n), mm(m), v(new T*[n])
{
    v[0] = new T[m*n];
    for (int i=1; i< n; i++)
        v[i] = v[i-1] + m;
}

template <class T>
NRMat<T>::NRMat(const T &a, int n, int m) : nn(n), mm(m), v(new T*[n])
{
    int i,j;
    v[0] = new T[m*n];
    for (i=1; i< n; i++)
        v[i] = v[i-1] + m;
    for (i=0; i< n; i++)
        for (j=0; j<m; j++)
            v[i][j] = a;
}

template <class T>
NRMat<T>::NRMat(const T *a, int n, int m) : nn(n), mm(m), v(new T*[n])

```

```

{
    int i,j;
    v[0] = new T[m*n];
    for (i=1; i< n; i++)
        v[i] = v[i-1] + m;
    for (i=0; i< n; i++)
        for (j=0; j<m; j++)
            v[i][j] = *a++;
}

template <class T>
NRMat<T>::NRMat(const NRMat &rhs) : nn(rhs.nn), mm(rhs.mm), v(new T*[nn])
{
    int i,j;
    v[0] = new T[mm*nn];
    for (i=1; i< nn; i++)
        v[i] = v[i-1] + mm;
    for (i=0; i< nn; i++)
        for (j=0; j<mm; j++)
            v[i][j] = rhs[i][j];
}

template <class T>
NRMat<T> & NRMat<T>::operator=(const NRMat<T> &rhs)
// postcondition: normal assignment via copying has been performed;
// if matrix and rhs were different sizes, matrix
// has been resized to match the size of rhs
{
    if (this != &rhs) {
        int i,j;
        if (nn != rhs.nn || mm != rhs.mm) {
            if (v != 0) {
                delete[] (v[0]);
                delete[] (v);
            }
            nn=rhs.nn;
            mm=rhs.mm;
            v = new T*[nn];
            v[0] = new T[mm*nn];
        }
        for (i=1; i< nn; i++)
            v[i] = v[i-1] + mm;
        for (i=0; i< nn; i++)
            for (j=0; j<mm; j++)
                v[i][j] = rhs[i][j];
    }
    return *this;
}

template <class T>
NRMat<T> & NRMat<T>::operator=(const T &a) //assign a to every element
{
    for (int i=0; i< nn; i++)
        for (int j=0; j<mm; j++)
            v[i][j] = a;
    return *this;
}

template <class T>
inline T* NRMat<T>::operator[](const int i) //subscripting: pointer to row i
{
    return v[i];
}

```

```

template <class T>
inline const T* NRMat<T>::operator[](const int i) const
{
    return v[i];
}

template <class T>
inline int NRMat<T>::nrows() const
{
    return nn;
}

template <class T>
inline int NRMat<T>::ncols() const
{
    return mm;
}

template <class T>
NRMat<T>::~NRMat()
{
    if (v != 0) {
        delete[] (v[0]);
        delete[] (v);
    }
}

template <class T>
class NRMat3d {
private:
    int nn;
    int mm;
    int kk;
    T ***v;
public:
    NRMat3d();
    NRMat3d(int n, int m, int k);
    inline T** operator[](const int i); //subscripting: pointer to row i
    inline const T* const * operator[](const int i) const;
    inline int dim1() const;
    inline int dim2() const;
    inline int dim3() const;
    ~NRMat3d();
};

template <class T>
NRMat3d<T>::NRMat3d(): nn(0), mm(0), kk(0), v(0) {}

template <class T>
NRMat3d<T>::NRMat3d(int n, int m, int k) : nn(n), mm(m), kk(k), v(new T**[n])
{
    int i,j;
    v[0] = new T*[n*m];
    v[0][0] = new T[n*m*k];
    for(j=1; j<m; j++)
        v[0][j] = v[0][j-1] + k;
    for(i=1; i<n; i++) {
        v[i] = v[i-1] + m;
        v[i][0] = v[i-1][0] + m*k;
        for(j=1; j<m; j++)
            v[i][j] = v[i][j-1] + k;
    }
}

```

```

template <class T>
inline T** NRMAT3d<T>::operator[](const int i) //subscripting: pointer to row i
{
    return v[i];
}

template <class T>
inline const T* const * NRMAT3d<T>::operator[](const int i) const
{
    return v[i];
}

template <class T>
inline int NRMAT3d<T>::dim1() const
{
    return nn;
}

template <class T>
inline int NRMAT3d<T>::dim2() const
{
    return mm;
}

template <class T>
inline int NRMAT3d<T>::dim3() const
{
    return kk;
}

template <class T>
NRMAT3d<T>::~~NRMAT3d()
{
    if (v != 0) {
        delete[] (v[0][0]);
        delete[] (v[0]);
        delete[] (v);
    }
}

//The next 3 classes are used in arithmetic coding, Huffman coding, and
//wavelet transforms respectively. This is as good a place as any to put them!

class arithcode {
private:
    NRVec<unsigned long> *ilob_p,*iupb_p,*ncumfq_p;
public:
    NRVec<unsigned long> &ilob,&iupb,&ncumfq;
    unsigned long jdif,nc,minint,nch,ncum,nrad;
    arithcode(unsigned long n1, unsigned long n2, unsigned long n3)
        : ilob_p(new NRVec<unsigned long>(n1)),
          iupb_p(new NRVec<unsigned long>(n2)),
          ncumfq_p(new NRVec<unsigned long>(n3)),
          ilob(*ilob_p),iupb(*iupb_p),ncumfq(*ncumfq_p) {}
    ~arithcode() {
        if (ilob_p != 0) delete ilob_p;
        if (iupb_p != 0) delete iupb_p;
        if (ncumfq_p != 0) delete ncumfq_p;
    }
};

class huffcode {
private:
    NRVec<unsigned long> *icod_p,*ncod_p,*left_p,*right_p;

```

```

public:
    NRVec<unsigned long> &icod,&ncod,&left,&right;
    int nch,nodemax;
    huffcode(unsigned long n1, unsigned long n2, unsigned long n3,
        unsigned long n4) :
        icod_p(new NRVec<unsigned long>(n1)),
        ncod_p(new NRVec<unsigned long>(n2)),
        left_p(new NRVec<unsigned long>(n3)),
        right_p(new NRVec<unsigned long>(n4)),
        icod(*icod_p),ncod(*ncod_p),left(*left_p),right(*right_p) {}
    ~huffcode() {
        if (icod_p != 0) delete icod_p;
        if (ncod_p != 0) delete ncod_p;
        if (left_p != 0) delete left_p;
        if (right_p != 0) delete right_p;
    }
};

class wavefilt {
private:
    NRVec<DP> *cc_p,*cr_p;
public:
    int ncof,ioff,joff;
    NRVec<DP> &cc,&cr;
    wavefilt() : cc(*cc_p),cr(*cr_p) {}
    wavefilt(const DP *a, const int n) : //initialize to array
        cc_p(new NRVec<DP>(n)),cr_p(new NRVec<DP>(n)),
        ncof(n),ioff(-(n >> 1)),joff(-(n >> 1)),cc(*cc_p),cr(*cr_p) {
        int i;
        for (i=0; i<n; i++)
            cc[i] = *a++;
        DP sig = -1.0;
        for (i=0; i<n; i++) {
            cr[n-1-i]=sig*cc[i];
            sig = -sig;
        }
    }
    ~wavefilt() {
        if (cc_p != 0) delete cc_p;
        if (cr_p != 0) delete cr_p;
    }
};

//Overloaded complex operations to handle mixed float and double
//This takes care of e.g. 1.0/z, z complex<float>

inline const complex<float> operator+(const double &a,
    const complex<float> &b) { return float(a)+b; }

inline const complex<float> operator+(const complex<float> &a,
    const double &b) { return a+float(b); }

inline const complex<float> operator-(const double &a,
    const complex<float> &b) { return float(a)-b; }

inline const complex<float> operator-(const complex<float> &a,
    const double &b) { return a-float(b); }

inline const complex<float> operator*(const double &a,
    const complex<float> &b) { return float(a)*b; }

inline const complex<float> operator*(const complex<float> &a,
    const double &b) { return a*float(b); }

```

```

inline const complex<float> operator/(const double &a,
    const complex<float> &b) { return float(a)/b; }

inline const complex<float> operator/(const complex<float> &a,
    const double &b) { return a/float(b); }

//some compilers choke on pow(float,double) in single precision. also atan2

inline float pow (float x, double y) {return pow(double(x),y);}
inline float pow (double x, float y) {return pow(x,double(y));}
inline float atan2 (float x, double y) {return atan2(double(x),y);}
inline float atan2 (double x, float y) {return atan2(x,double(y));}
#endif /* _NR_UTIL_H_ */

```

If you want to use a different matrix/vector class library than the above default, you need to do two things. The first is to change the constness of the type declarations, as discussed in §1.3. Here is a listing of the file `nrtypes_lib.h`. You use it by changing the file that is included by `nrtypes.h` from `nrtypes_nr.h` to `nrtypes_lib.h`.

```

#ifndef _NR_TYPES_H_
#define _NR_TYPES_H_

#include <complex>
#include <fstream>
#include "nrutil.h"
using namespace std;

typedef double DP;

// Vector Types

typedef const NRVec<bool> Vec_I_BOOL;
typedef const NRVec<bool> Vec_BOOL, Vec_O_BOOL, Vec_IO_BOOL;

typedef const NRVec<char> Vec_I_CHR;
typedef const NRVec<char> Vec_CHR, Vec_O_CHR, Vec_IO_CHR;

typedef const NRVec<unsigned char> Vec_I_UCHR;
typedef const NRVec<unsigned char> Vec_UCHR, Vec_O_UCHR, Vec_IO_UCHR;

typedef const NRVec<int> Vec_I_INT;
typedef const NRVec<int> Vec_INT, Vec_O_INT, Vec_IO_INT;

typedef const NRVec<unsigned int> Vec_I_UINT;
typedef const NRVec<unsigned int> Vec_UINT, Vec_O_UINT, Vec_IO_UINT;

typedef const NRVec<long> Vec_I_LNG;
typedef const NRVec<long> Vec_LNG, Vec_O_LNG, Vec_IO_LNG;

typedef const NRVec<unsigned long> Vec_I_ULNG;
typedef const NRVec<unsigned long> Vec_ULNG, Vec_O_ULNG, Vec_IO_ULNG;

typedef const NRVec<float> Vec_I_SP;
typedef const NRVec<float> Vec_SP, Vec_O_SP, Vec_IO_SP;

typedef const NRVec<DP> Vec_I_DP;
typedef const NRVec<DP> Vec_DP, Vec_O_DP, Vec_IO_DP;

typedef const NRVec<complex<float> > Vec_I_CPLX_SP;
typedef const NRVec<complex<float> > Vec_CPLX_SP, Vec_O_CPLX_SP, Vec_IO_CPLX_SP;

```

```

typedef const NRVec<complex<DP> > Vec_I_CPLX_DP;
typedef const NRVec<complex<DP> > Vec_CPLX_DP, Vec_O_CPLX_DP, Vec_IO_CPLX_DP;

// Matrix Types

typedef const NRMat<bool> Mat_I_BOOL;
typedef const NRMat<bool> Mat_BOOL, Mat_O_BOOL, Mat_IO_BOOL;

typedef const NRMat<char> Mat_I_CHR;
typedef const NRMat<char> Mat_CHR, Mat_O_CHR, Mat_IO_CHR;

typedef const NRMat<unsigned char> Mat_I_UCHR;
typedef const NRMat<unsigned char> Mat_UCHR, Mat_O_UCHR, Mat_IO_UCHR;

typedef const NRMat<int> Mat_I_INT;
typedef const NRMat<int> Mat_INT, Mat_O_INT, Mat_IO_INT;

typedef const NRMat<unsigned int> Mat_I_UINT;
typedef const NRMat<unsigned int> Mat_UINT, Mat_O_UINT, Mat_IO_UINT;

typedef const NRMat<long> Mat_I_LNG;
typedef const NRMat<long> Mat_LNG, Mat_O_LNG, Mat_IO_LNG;

typedef const NRVec<unsigned long> Mat_I_ULNG;
typedef const NRMat<unsigned long> Mat_ULNG, Mat_O_ULNG, Mat_IO_ULNG;

typedef const NRMat<float> Mat_I_SP;
typedef const NRMat<float> Mat_SP, Mat_O_SP, Mat_IO_SP;

typedef const NRMat<DP> Mat_I_DP;
typedef const NRMat<DP> Mat_DP, Mat_O_DP, Mat_IO_DP;

typedef const NRMat<complex<float> > Mat_I_CPLX_SP;
typedef const NRMat<complex<float> > Mat_CPLX_SP, Mat_O_CPLX_SP, Mat_IO_CPLX_SP;

typedef const NRMat<complex<DP> > Mat_I_CPLX_DP;
typedef const NRMat<complex<DP> > Mat_CPLX_DP, Mat_O_CPLX_DP, Mat_IO_CPLX_DP;

// 3D Matrix Types

typedef const NRMat3d<DP> Mat3D_I_DP;
typedef NRMat3d<DP> Mat3D_DP, Mat3D_O_DP, Mat3D_IO_DP;

// Miscellaneous Types

typedef NRVec<unsigned long *> Vec_ULNG_p;
typedef const NRVec<const NRMat<DP> *> Vec_Mat_DP_p;
typedef NRVec<fstream *> Vec_FSTREAM_p;

#endif /* _NR_TYPES_H */

```

The second thing you need to do to use a different matrix/vector library is to supply a wrapper-class replacement for `nrutil.h`. Here, for example, we give the listing of the file `nrutil_tnt.h` that enables you to use the TNT matrix/vector library [1] instead of our default one. You use it by changing the file that is included by `nrutil.h` from `nrutil_nr.h` to `nrutil_tnt.h`.

```

#ifndef _NR_UTIL_H_
#define _NR_UTIL_H_

#include <string>
#include <cmath>
#include <complex>

```

```

#include <iostream>
using namespace std;

typedef double DP;

template<class T>
inline const T SQR(const T a) {return a*a;}

template<class T>
inline const T MAX(const T &a, const T &b)
{return b > a ? (b) : (a);}

inline float MAX(const double &a, const float &b)
{return b > a ? (b) : float(a);}

inline float MAX(const float &a, const double &b)
{return b > a ? float(b) : (a);}

template<class T>
inline const T MIN(const T &a, const T &b)
{return b < a ? (b) : (a);}

inline float MIN(const double &a, const float &b)
{return b < a ? (b) : float(a);}

inline float MIN(const float &a, const double &b)
{return b < a ? float(b) : (a);}

template<class T>
inline const T SIGN(const T &a, const T &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

inline float SIGN(const float &a, const double &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

inline float SIGN(const double &a, const float &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

template<class T>
inline void SWAP(T &a, T &b)
{T dum=a; a=b; b=dum;}

namespace NR {
  inline void nrerror(const string error_text)
  // Numerical Recipes standard error handler
  {
    cerr << "Numerical Recipes run-time error..." << endl;
    cerr << error_text << endl;
    cerr << "...now exiting to system..." << endl;
    exit(1);
  }
}

#include "tnt/tnt.h"
#include "tnt/vec.h"
#include "tnt/cmat.h"

// TNT Wrapper File
// This is the file that "joins" the TNT Vector<> and Matrix<> classes
// to the NRVec and NRMat classes by the Wrapper Class Method

// NRVec contains a Vector and a &Vector. All its constructors, except the
// conversion constructor, create the Vector and point the &Vector to it.
// The conversion constructor only points the &Vector. All operations

```

```

// (size, subscript) are through the &Vector, which as a reference
// (not pointer) has no indirection overhead.

template<class T>
class NRVec {
private:
    TNT::Vector<T> myvec;
    TNT::Vector<T> &myref;
public:
    NRVec<T>() : myvec(), myref(myvec) {}
    explicit NRVec<T>(const int n) : myvec(n), myref(myvec) {}
    NRVec<T>(const T &a, int n) : myvec(n,a), myref(myvec) {}
    NRVec<T>(const T *a, int n) : myvec(n,a), myref(myvec) {}
    NRVec<T>(TNT::Vector<T> &rhs) : myref(rhs) {}
    // conversion constructor makes a special NRVec pointing to Vector's data
    // this handles Vector actual args sent to NRVec formal args in functions
    NRVec(const NRVec<T>& rhs) : myvec(rhs.myref), myref(myvec) {}
    // copy constructor calls Vector copy constructor
    inline NRVec& operator=(const NRVec& rhs) { myref=rhs.myref; return *this;}
    // assignment operator calls Vector assignment operator
    inline NRVec& operator=(const T& rhs) { myvec=rhs; return *this;}
    // scalar assignment calls Vector assignment operator
    inline int size() const {return myref.size();}
    inline T & operator[](const int i) const {return myref[i];}
    // return element i
    inline operator TNT::Vector<T>() const {return myref;}
    // conversion operator to Vector
    // this handles NRVec function return types when used in Vector expressions
    ~NRVec() {}
};

template <class T>
class NRMAT {
private:
    TNT::Matrix<T> mymat;
    TNT::Matrix<T> &myref;
public:
    NRMAT() : mymat(), myref(mymat) {}
    NRMAT(int n, int m) : mymat(n,m), myref(mymat) {}
    NRMAT(const T& a, int n, int m) : mymat(n,m,a), myref(mymat) {}
    //Initialize to constant
    NRMAT(const T* a, int n, int m) : mymat(n,m,a), myref(mymat) {}
    //Initialize to array
    NRMAT<T>(TNT::Matrix<T> &rhs) : myref(rhs) {}
    // conversion constructor from Matrix
    NRMAT(const NRMAT& rhs) : mymat(rhs.myref), myref(mymat) {}
    // copy constructor
    inline NRMAT& operator=(const NRMAT& rhs) { myref=rhs.myref; return *this;}
    // assignment operator
    inline NRMAT& operator=(const T& rhs) { mymat=rhs; return *this;}
    // scalar assignment calls Matrix assignment operator
    inline T* operator[](const int i) const {return myref[i];}
    //subscripting: pointer to row i
    //return type is whatever Matrix returns for a single [] dereference
    inline int nrows() const {return myref.num_rows();}
    inline int ncols() const {return myref.num_cols();}
    inline operator TNT::Matrix<T>() const {return myref;}
    // conversion operator to Matrix
    ~NRMAT() {}
};

template <class T>
class NRMAT3d {
The rest of the code is identical to nrutil_nr.h and is omitted.

```

Finally we give the listing of the file `nrutil_mtl.h` that enables you to use the MTL matrix/vector library [2]. Again, simply change the file that is included by `nrutil.h` from `nrutil_nr.h` to `nrutil_mtl.h`.

Note that the MTL library uses the `std::vector` class to define its basic vector type. This class has a specialization for `vector<bool>` that doesn't work with our wrapper class scheme. So we implement `NRVec<bool>` as our own specialization of `NRVec` by making it a derived class of `NRVec<int>`. (Recall that a `bool` variable is automatically converted to `int` when necessary.) The only potential problem with doing this occurs if you blindly try to mix objects of type `mtl::Vector<bool>` with objects of type `NRVec<bool>`!

```
#ifndef _NR_UTIL_H_
#define _NR_UTIL_H_

#include <string>
#include <cmath>
#include <complex>
#include <iostream>
using namespace std;

typedef double DP;

template<class T>
inline const T SQR(const T a) {return a*a;}

template<class T>
inline const T MAX(const T &a, const T &b)
{return b > a ? (b) : (a);}

inline float MAX(const double &a, const float &b)
{return b > a ? (b) : float(a);}

inline float MAX(const float &a, const double &b)
{return b > a ? float(b) : (a);}

template<class T>
inline const T MIN(const T &a, const T &b)
{return b < a ? (b) : (a);}

inline float MIN(const double &a, const float &b)
{return b < a ? (b) : float(a);}

inline float MIN(const float &a, const double &b)
{return b < a ? float(b) : (a);}

template<class T>
inline const T SIGN(const T &a, const T &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

inline float SIGN(const float &a, const double &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

inline float SIGN(const double &a, const float &b)
{return b >= 0 ? (a >= 0 ? a : -a) : (a >= 0 ? -a : a);}

template<class T>
inline void SWAP(T &a, T &b)
{T dum=a; a=b; b=dum;}

namespace NR {
    inline void nrerror(const string error_text)
```

```

// Numerical Recipes standard error handler
{
    cerr << "Numerical Recipes run-time error..." << endl;
    cerr << error_text << endl;
    cerr << "...now exiting to system..." << endl;
    exit(1);
}
}

#include "mtl/mtl.h"
using namespace mtl;

// mtl Wrapper File
// This is the file that "joins" the mtl Vector<> and Matrix<> classes
// to the NRVec and NRMat classes by the Wrapper Class Method

// NRVec contains a Vector and a &Vector. All its constructors, except the
// conversion constructor, create the Vector and point the &Vector to it.
// The conversion constructor only points the &Vector. All operations
// (size, subscript) are through the &Vector, which as a reference
// (not pointer) has no indirection overhead.

template<class T>
class NRVec {
// Use the std::vector based dense1D for our Vector type
typedef dense1D<T> Vector;
protected:    //access required in NRVec<bool> below
    Vector myvec;
    Vector &myref;
public:
    NRVec<T>() : myvec(), myref(myvec) {}
    explicit NRVec<T>(const int n) : myvec(n), myref(myvec) {}
    NRVec<T>(const T &a, int n) : myvec(n), myref(myvec) {
        for (int i=0; i<n; i++) myvec[i] = a;}
    NRVec<T>(const T *a, int n) : myvec(n), myref(myvec) {
        for (int i=0; i<n; i++) myvec[i] = *a++;}
    NRVec<T>(Vector &rhs) : myref(rhs) {}
    // conversion constructor makes a special NRVec pointing to Vector's data
    // this handles Vector actual args sent to NRVec formal args in functions
    NRVec(const NRVec<T>& rhs) : myvec(rhs.myref.size()), myref(myvec)
        {copy(rhs.myref,myref);}
    // copy constructor. mtl copy constructor
    // does shallow copy only. so use copy() instead
    inline NRVec& operator=(const NRVec& rhs) {
        if (myref.size() != rhs.myref.size())
            myref.resize(rhs.myref.size());
        copy(rhs.myref,myref); return *this;}
    inline int size() const {return myref.size();}
    inline T & operator[](const int i) const {return myref[i];}
    inline operator Vector() const {return myref;}
    // conversion operator to Vector
    // handles NRVec function return types when used in Vector expressions
    ~NRVec() {}
};

//The std:vector class has a specialization for vector<bool> that doesn't
//work with the above wrapper class scheme. So implement our own
//specialization as a derived class of NRVec<int>. This could cause
//problems if you mix mtl::Vector<bool> with NRVec<bool>!

template <> class NRVec<bool> : public NRVec<int> {
public:
    NRVec() : NRVec<int>() {}
    explicit NRVec(const int n) : NRVec<int>(n) {}
};

```

```

    NRVec(const bool &a, int n) : NRVec<int>(int(a),n) {}
    NRVec(const bool *a, int n) : NRVec<int>(n) {
        for (int i=0; i<n; i++) myvec[i] = *a++;}
//note: defaults OK for copy constructor and assignment
};

template <class T>
class NRMat {
// Use the matrix generator to select a matrix type
typedef matrix< T,
    rectangle<>,
    dense<>,
    row_major>::type Matrix;
protected:
    Matrix mymat;
    Matrix &myref;
public:
    NRMat() : mymat(), myref(mymat) {}
    NRMat(int n, int m) : mymat(n,m), myref(mymat) {}
    NRMat(const T& a, int n, int m) : mymat(n,m), myref(mymat) {
        for (int i=0; i<n; i++)
            for (int j=0; j<m; j++)
                mymat[i][j] = a;}
    NRMat(const T* a, int n, int m) : mymat(n,m), myref(mymat) {
        for (int i=0; i<n; i++)
            for (int j=0; j<m; j++)
                mymat[i][j] = *a++;}
    NRMat<T>(Matrix &rhs) : myref(rhs) {}
    NRMat(const NRMat& rhs) :
        mymat(rhs.myref.nrows(),rhs.myref.ncols()), myref(mymat)
    {copy(rhs.myref,myref);}
    inline NRMat& operator=(const NRMat& rhs) {
        if (myref.nrows() != rhs.myref.nrows() && myref.ncols() !=
            rhs.myref.ncols()) {
            cerr << "assignment with incompatible matrix sizes\n";
            abort();
        }
        copy(rhs.myref,myref); return *this;
    }
    typename Matrix::OneD operator[](const int i) const {return myref[i];}
//return type is whatever Matrix returns for a single [] dereference
    inline int nrows() const {return myref.nrows();}
    inline int ncols() const {return myref.ncols();}
    inline operator Matrix() const {return myref;}
    ~NRMat() {}
};

template <> class NRMat<bool> : public NRMat<int> {
public:
    NRMat() : NRMat<int>() {}
    explicit NRMat(int n, int m) : NRMat<int>(n,m) {}
    NRMat(const bool &a, int n, int m) : NRMat<int>(int(a),n,m) {}
    NRMat(const bool *a, int n, int m) : NRMat<int>(n,m) {
        for (int i=0; i<n; i++)
            for (int j=0; j<m; j++)
                mymat[i][j] = *a++;}
//note: defaults OK for copy constructor and assignment
};

template <class T>
class NRMat3d {
The rest of the code is identical to nrutil_nr.h and is omitted.

```

CITED REFERENCES AND FURTHER READING:

Pozo, R., *Template Numerical Toolkit*, <http://math.nist.gov/tnt>. [1]

Lumsdaine, A., and Siek, J. 1998, *The Matrix Template Library*, <http://www.lsc.nd.edu/research/mtl>. [2]