```
   Help
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <math.h>
#include <string.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(floa
    t)));
  if (!v) nrerror("allocation failure in vector()");

  memset( (char *)v, '{0', ((nh-nl+1+NR_END)*sizeof(float)
    ) );
  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)*si
    zeof(unsigned char)));
  if (!v) nrerror("allocation failure in cvector()");

  memset( (char *)v, '{0', ((nh-nl+1+NR_END)*sizeof(unsign
    ed char)) );
  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)*si
    zeof(long)));
  if (!v) nrerror("allocation failure in lvector()");

  memset( (char *)v, '{0', ((nh-nl+1+NR_END)*sizeof(long))
     );
  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()");

  memset( (char *)v, '{0', ((nh-nl+1+NR_END)*sizeof(
    double)) );
  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()");

  memset( (char *)m, '{0', ((nrow+NR_END)*sizeof(float*))
    );

  m += NR_END;
  m -= nrl;

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

  memset( (char *)m[nrl], '{0', ((nrow*ncol+NR_END)*sizeof
    (float)) );

  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)*si
    zeof(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(floa
    t*)));
  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 po
    ints 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(floa
    t*)));
  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..nc
    h][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(floa
    t**)));
  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)*si
    zeof(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 f3ten
    sor()");
  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 nc
    l, 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, lon
    g 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 nc
    l, 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));
}

#endif //PremiaCurrentVersion
```

References