C. jacobnbndf

Calculates the Jacobian matrix of an n-dimensional function of n variables, if this Jacobian is known to be a band matrix and have to be stored rowwise in a one-dimensional array. *jacobnbndf* computes first order difference quotient approximations

$$J_{i,j} = (f_i(x_1,...,x_{j-1},x_j+\delta_i,x_{j+1},...,x_n)-f_i(x_1,...,x_{j-1},x_j,x_{j+1},...,x_n))/\delta i$$

for $i=1,...n$; $\max(1,i\text{-}lw)\leq j\leq \min(n,i+rw)$ to the partial derivatives $J_{i,j} = \partial f_i(x)/\partial x_j$ of the components of the function $f(x)$ $(f,x\epsilon R^n)$.

Function Parameters:

void jacobnbndf (*n,lw,rw,x,f,jac,di,funct*)

n: int;
  entry: the number of independent variables and the dimension of the function;

lw: int;
  entry: the number of codiagonals to the left of the main diagonal of the Jacobian matrix, which is known to be a band matrix;

rw: int;
  entry: the number of codiagonals to the right of the main diagonal of the Jacobian matrix;

x: float *x[1:n]*;
  entry: the point at which the Jacobian has to be calculated;

f: float *f[1:n]*;
  entry: the values of the function components at the point given in array *x*;

jac: float *jac[1:(lw+rw)*(n-1)+n]*;
  exit: the Jacobian matrix in such a way that the (i,j)-th element of the Jacobian, i.e. the partial derivative of *f[i]* to *x[j]* is given in *jac[(lw+rw)*(i-1)+j]*, $i=1,...,n$, $j=\max(1,i\text{-}lw),...,\min(n,i+rw)$;

di: float *(*di)(i), int i*;
  entry: the partial derivatives to *x[i]* are approximated with forward differences, using an increment to the i-th variable that equals the value of *di*, $i=1,...,n$;

funct: void *(*funct)(n,l,u,x,f)*;
  entry: the meaning of the parameters of the function *funct* is as follows:
    n: the number of function components;
    l,u: int; the lower and upper bound of the function component subscript;
    x: the independent variables are given in *x[1:n]*;
    f: after a call of *funct* the function components *f[i]*, $i=l,...,u$, should be given in *f[l:u]*.

```
void jacobnbndf(int n, int lw, int rw, float x[], float f[],
                  float jac[], float (*di)(int),
                  int (*funct)(int, int, int, float[], float[]))
{
   float *allocate_real_vector(int, int);
   void free_real_vector(float *, int);
   int i,j,k,l,u,t,b,ll;
   float aid,stepi,*f1;

   l=1;
   u=lw+1;
   t=rw+1;
   b=lw+rw;
   for (i=1; i<=n; i++) {
      ll=l;
```

```
        f1=allocate_real_vector(ll,u);
        stepi=(*di)(i);
        aid=x[i];
        x[i]=aid+stepi;
        (*funct)(n,l,u,x,f1);
        x[i]=aid;
        k = i+((i <= t) ? 0 : i-t)*b;
        for (j=l; j<=u; j++) {
            jac[k]=(f1[j]-f[j])/stepi;
            k += b;
        }
        if (i >= t) l++;
        if (u < n) u++;
        free_real_vector(f1,ll);
    }
}
```