NUMERICAL RECIPES

Webnote No. 27, Rev. 1

Code for Machar

machar.h

```
struct Machar {
    Int ibeta, it, irnd, ngrd, machep, negep, iexp, minexp, maxexp;
    Doub eps, epsneg, xmin, xmax;

    Machar () {
```

Determines machine-specific parameters affecting floating-point arithmetic, including `ibeta`, the floating-point radix; `it`, the number of base-`ibeta` digits in the floating-point mantissa; `eps`, the smallest positive number that, added to 1.0, is not equal to 1.0; `epsneg`, the smallest positive number that, subtracted from 1.0, is not equal to 1.0; `xmin`, the smallest representable positive number; and `xmax`, the largest representable positive number. See text for description of other returned parameters. Change `Doub` to `float` to find single precision parameters.

```
        Int i,itemp,iz,j,k,mx,nxres;
        Doub a,b,beta,betah,betain,one,t,temp,temp1,tempa,two,y,z,zero;
        one=Doub(1);
        two=one+one;
        zero=one-one;
        a=one;                          Determine ibeta and beta by the method of M.
        do {                                Malcolm.
            a += a;
            temp=a+one;
            temp1=temp-a;
        } while (temp1-one == zero);
        b=one;
        do {
            b += b;
            temp=a+b;
            itemp=Int(temp-a);
        } while (itemp == 0);
        ibeta=itemp;
        beta=Doub(ibeta);
        it=0;                           Determine it and irnd.
        b=one;
        do {
            ++it;
            b *= beta;
            temp=b+one;
            temp1=temp-b;
        } while (temp1-one == zero);
        irnd=0;
        betah=beta/two;
        temp=a+betah;
        if (temp-a != zero) irnd=1;
        tempa=a+beta;
        temp=tempa+betah;
        if (irnd == 0 && temp-tempa != zero) irnd=2;
        negep=it+3;                     Determine negep and epsneg.
        betain=one/beta;
```


Copyright 2007 Numerical Recipes Software

```
a=one;
for (i=1;i<=negep;i++) a *= betain;
b=a;
for (;;) {
    temp=one-a;
    if (temp-one != zero) break;
    a *= beta;
    --negep;
}
negep = -negep;
epsneg=a;
machep = -it-3;                 Determine machep and eps.
a=b;
for (;;) {
    temp=one+a;
    if (temp-one != zero) break;
    a *= beta;
    ++machep;
}
eps=a;
ngrd=0;                         Determine ngrd.
temp=one+eps;
if (irnd == 0 && temp*one-one != zero) ngrd=1;
i=0;                            Determine iexp.
k=1;
z=betain;
t=one+eps;
nxres=0;
for (;;) {                      Loop until an underflow occurs, then exit.
    y=z;
    z=y*y;
    a=z*one;                    Check here for the underflow.
    temp=z*t;
    if (a+a == zero || abs(z) >= y) break;
    temp1=temp*betain;
    if (temp1*beta == z) break;
    ++i;
    k += k;
}
if (ibeta != 10) {
    iexp=i+1;
    mx=k+k;
} else {                        For decimal machines only.
    iexp=2;
    iz=ibeta;
    while (k >= iz) {
        iz *= ibeta;
        ++iexp;
    }
    mx=iz+iz-1;
}
for (;;) {                      To determine minexp and xmin, loop until an
    xmin=y;                         underflow occurs, then exit.
    y *= betain;
    a=y*one;                    Check here for the underflow.
    temp=y*t;
    if (a+a != zero && abs(y) < xmin) {
        ++k;
        temp1=temp*betain;
        if (temp1*beta == y && temp != y) {
            nxres=3;
            xmin=y;
            break;
        }
```

Copyright 2007 Numerical Recipes Software

```
            }
            else break;
        }
        minexp = -k;                    Determine maxexp, xmax.
        if (mx <= k+k-3 && ibeta != 10) {
            mx += mx;
            ++iexp;
        }
        maxexp=mx+minexp;
        irnd += nxres;                  Adjust irnd to reflect partial underflow.
        if (irnd >= 2) maxexp -= 2;     Adjust for IEEE-style machines.
        i=maxexp+minexp;
        Adjust for machines with implicit leading bit in binary mantissa, and machines with
        radix point at extreme right of mantissa.
        if (ibeta == 2 && !i) --maxexp;
        if (i > 20) --maxexp;
        if (a != y) maxexp -= 2;
        xmax=one-epsneg;
        if (xmax*one != xmax) xmax=one-beta*epsneg;
        xmax /= (xmin*beta*beta*beta);
        i=maxexp+minexp+3;
        for (j=1;j<=i;j++) {
            if (ibeta == 2) xmax += xmax;
            else xmax *= beta;
        }
    }

    void report() {
        cout << "quantity:  numeric_limits<Doub> says  (we calculate)" << endl;
        cout << "radix:  " << numeric_limits<Doub>::radix
            << "  (" << ibeta << ")" << endl;
        cout << "mantissa digits:  " << numeric_limits<Doub>::digits
            << "  (" << it << ")" << endl;
        cout << "round style:  " << numeric_limits<Doub>::round_style
            << "  (" << irnd << ") [our 5 == IEEE 1]" << endl;
        cout << "guard digits:  " << "[not in numeric_limits]"
            << "  (" << ngrd << ")" << endl;
        cout << "epsilon:  " << numeric_limits<Doub>::epsilon()
            << "  (" << eps << ")" << endl;
        cout << "neg epsilon:  " << "[not in numeric_limits]"
            << "  (" << epsneg << ")" << endl;
        cout << "epsilon power:  " << "[not in numeric_limits]"
            << "  (" << machep << ")" << endl;
        cout << "neg epsilon power:  " << "[not in numeric_limits]"
            << "  (" << negep << ")" << endl;
        cout << "exponent digits:  " << "[not in numeric_limits]"
            << "  (" << iexp << ")" << endl;
        cout << "min exponent:  " << numeric_limits<Doub>::min_exponent
            << "  (" << minexp << ")" << endl;
        cout << "max exponent:  " << numeric_limits<Doub>::max_exponent
            << "  (" << maxexp << ")" << endl;
        cout << "minimum:  " << numeric_limits<Doub>::min()
            << "  (" << xmin << ")" << endl;
        cout << "maximum:  " << numeric_limits<Doub>::max()
            << "  (" << xmax << ")" << endl;
    }
};
```

Copyright 2007 Numerical Recipes Software