NUMERICAL RECIPES

Webnote No. 4, Rev. 1

Implementation of Adapt

adapt.h

```
struct Adapt {
```
Adaptive quadrature for the integral of the function `func` from `a` to `b`. Integration is performed by a Gauss-Lobatto method with a Kronrod extension.
```
    Doub TOL,toler;
    static const Doub alpha,beta,x1,x2,x3,x[12];   Abscissas for Gauss-Lobatto-Kronrod
    bool terminate,out_of_tolerance;                    quadrature.
    Adapt(Doub tol);
    template <class T>
    Doub integrate(T &func, const Doub a, const Doub b);
    template <class T>
    Doub adaptlob(T &func, const Doub a, const Doub b, const Doub fa,
        const Doub fb, const Doub is);
};

Adapt::Adapt(Doub tol) : TOL(tol),terminate(true),out_of_tolerance(false)
```
Constructor is invoked with desired tolerance `tol`. The smallest allowed value of `tol` is `10*EPS`, where `EPS` is the machine precision. If `tol` is input as less than this (e.g. `tol` $= 0$), then `tol` is set to `10*EPS`.
```
{
    const Doub EPS=numeric_limits<Doub>::epsilon();
    if (TOL < 10.0*EPS)
        TOL=10.0*EPS;
}

template <class T>
Doub Adapt::integrate(T &func, const Doub a, const Doub b)
{
    Doub m,h,fa,fb,i1,i2,is,erri1,erri2,r,y[13];
    m=0.5*(a+b);
    h=0.5*(b-a);
    fa=y[0]=func(a);
    fb=y[12]=func(b);
    for (Int i=1;i<12;i++)
        y[i]=func(m+x[i]*h);
    i2=(h/6.0)*(y[0]+y[12]+5.0*(y[4]+y[8]));        4-point Gauss-Lobatto formula.
    i1=(h/1470.0)*(77.0*(y[0]+y[12])+432.0*(y[2]+y[10])+
        625.0*(y[4]+y[8])+672.0*y[6]);              7-point Kronrod extension.
    is=h*(0.0158271919734802*(y[0]+y[12])+0.0942738402188500*
        (y[1]+y[11])+0.155071987336585*(y[2]+y[10])+
        0.188821573960182*(y[3]+y[9])+0.199773405226859*
        (y[4]+y[8])+0.224926465333340*(y[5]+y[7])+
        0.242611071901408*y[6]);                    13-point Kronrod extension.
    erri1=abs(i1-is);
    erri2=abs(i2-is);
    r=(erri2 != 0.0) ? erri1/erri2 : 1.0;
    toler=(r > 0.0 && r < 1.0) ? TOL/r : TOL;       Error of i1 will be sufficiently small
    if (is == 0.0)                                      that we can increase tolerance.
        is=b-a;
```

```
    is=abs(is);
    return adaptlob(func,a,b,fa,fb,is);
}

template <class T>
Doub Adapt::adaptlob(T &func, const Doub a, const Doub b, const Doub fa,
        const Doub fb, const Doub is)
```

Helper function for recursion.

```
{
    Doub m,h,mll,ml,mr,mrr,fmll,fml,fm,fmrr,fmr,i1,i2;
    m=0.5*(a+b);
    h=0.5*(b-a);
    mll=m-alpha*h;
    ml=m-beta*h;
    mr=m+beta*h;
    mrr=m+alpha*h;
    fmll=func(mll);
    fml=func(ml);
    fm=func(m);
    fmr=func(mr);
    fmrr=func(mrr);
    i2=h/6.0*(fa+fb+5.0*(fml+fmr));                    4-point Gauss-Lobatto formula.
    i1=h/1470.0*(77.0*(fa+fb)+432.0*(fmll+fmrr)+625.0*(fml+fmr)+672.0*fm);
```

7-point Kronrod extension.

```
    if (abs(i1-i2) <= toler*is || mll <= a || b <= mrr) {
        if ((mll <= a || b <= mrr) && terminate) {
            out_of_tolerance=true;                     Interval contains no more machine
            terminate=false;                               numbers.
        }
        return i1;                                     Terminate recursion.
    }
    else                                               Subdivide interval.
        return adaptlob(func,a,mll,fa,fmll,is)+
            adaptlob(func,mll,ml,fmll,fml,is)+
            adaptlob(func,ml,m,fml,fm,is)+
            adaptlob(func,m,mr,fm,fmr,is)+
            adaptlob(func,mr,mrr,fmr,fmrr,is)+
            adaptlob(func,mrr,b,fmrr,fb,is);
}
const Doub Adapt::alpha=sqrt(2.0/3.0);
const Doub Adapt::beta=1.0/sqrt(5.0);
const Doub Adapt::x1=0.942882415695480;
const Doub Adapt::x2=0.641853342345781;
const Doub Adapt::x3=0.236383199662150;
const Doub Adapt::x[12]={0,-x1,-alpha,-x2,-beta,-x3,0.0,x3,beta,x2,alpha,x1};
```