NUMERICAL RECIPES

Webnote No. 22, Rev. 1

Dense Output for Stoermer's Rule

The dense output routines are very similar to those of `StepperBS`. However, since y, y' and y'' are available instead of simply y and y', the details are different. Our coding follows closely that of the Fortran routine ODEX2 [1]

stepperstoerm.h

```
template <class D>
void StepperStoerm<D>::prepare_dense(const Doub h,VecDoub_I &dydxnew,
    VecDoub_I &ysav,VecDoub_I &scale,const Int k, Doub &error) {
```

Store coefficients of interpolating polynomial for dense output in `dens` array. Input stepsize `h`, right-hand side at end of interval `dydxnew` (only first `n/2` elements referenced), y and y' at beginning of interval in `ysav[0..2n-1]`, scale factor `atol` $+ (|y|, |y'|)$`rtol` in `scale[0..2n-1]`, and column `k` in which convergence was achieved. Output interpolation error in `error`.

```
    Doub h2=h*h;
    mu=MAX(1,2*k-3);                        Degree of interpolating polynomial is mu+6.
    for (Int i=0; i<neqn; i++) {            Store y, y' and y'' at both ends of interval.
        dens[i]=ysav[i];
        dens[neqn+i]=h*ysav[neqn+i];
        dens[2*neqn+i]=h2*dydx[i];
        dens[3*neqn+i]=y[i];
        dens[4*neqn+i]=h*y[neqn+i];
        dens[5*neqn+i]=h2*dydxnew[i];
    }
    for (Int j=1; j<=k; j++) {              Compute y and y' at midpoint.
        Doub dblenj=nseq[j];
        for (Int l=j; l>=1; l--) {
            Doub factor=SQR(dblenj/nseq[l-1])-1.0;
            for (Int i=0; i<neqn; i++) {
                ysave[l-1][i]=ysave[l][i]+(ysave[l][i]-ysave[l-1][i])/factor;
                ysavep[l-1][i]=ysavep[l][i]+(ysavep[l][i]-ysavep[l-1][i])/factor;
            }
        }
    }
    for (Int i=0; i<neqn; i++) {
        dens[6*neqn+i]=ysave[0][i];
        dens[7*neqn+i]=h*ysavep[0][i];
    }
    for (Int kmi=2; kmi<=mu; kmi++) {       Compute kmi-th derivative at midpoint.
        Int kbeg=(kmi-2)/2;
        if (kmi == 2*kbeg+2) {
            if (kmi == 2) {
                for (Int i=0; i<neqn; i++)
                    ysave[0][i]=0.5*(dydxnew[i]+fsave[0][i]);
                kbeg=1;
            }
            for (Int kk=kbeg; kk<=k; kk++) {
                Doub facnj=0.5*pow(nseq[kk]/2.0,kmi-2);
                Int ipt=kk*kk+kk+kmi/2-2;
                for (Int i=0; i<neqn; i++)
                    ysave[kk][i]=(fsave[ipt][i]+fsave[ipt+1][i])*facnj;
```

```
            }
        } else {
            for (Int kk=kbeg; kk<=k; kk++) {
                Doub facnj=pow(nseq[kk]/2.0,kmi-2);
                Int ipt=kk*kk+kk+kbeg;
                for (Int i=0; i<neqn; i++)
                    ysave[kk][i]=fsave[ipt][i]*facnj;
            }
        }
        for (Int j=kbeg+1; j<=k; j++) {          Extrapolation.
            Doub dblenj=nseq[j];
            for (Int l=j; l>=kbeg+1; l--) {
                Doub factor=SQR(dblenj/nseq[l-1])-1.0;
                for (Int i=0; i<neqn; i++)
                    ysave[l-1][i]=ysave[l][i]+
                        (ysave[l][i]-ysave[l-1][i])/factor;
            }
        }
        for (Int i=0; i<neqn; i++)
            dens[(kmi+6)*neqn+i]=ysave[kbeg][i]*h2;
        if (kmi == mu) continue;
        for (Int kk=(kmi-1)/2; kk<=k; kk++) {    Compute differences.
            Int lbeg=kk*kk+kmi-2;
            Int lend=SQR(kk+1)-1;
            if (kmi == 2) lbeg++;
            for (Int l=lend; l>=lbeg; l--)
                for (Int i=0; i<neqn; i++)
                    fsave[l][i]=fsave[l][i]-fsave[l-1][i];
            if (kmi == 2) {
                Int l=lbeg-1;
                for (Int i=0; i<neqn; i++)
                    fsave[l][i]=fsave[l][i]-dydx[i];
            }
        }
    }
    dense_interp(neqn,dens,mu);              Compute the interpolation coefficients in dens.
    error=0.0;                               Estimate the interpolation error.
    if (mu >= 1) {
        for (Int i=0; i<neqn; i++)
            error += SQR(dens[(mu+6)*neqn+i]/scale[i]);
        error=sqrt(error/neqn)*errfac[mu-1];
    }
}
template <class D>
Doub StepperStoerm<D>::dense_out(const Int i,const Doub x,const Doub h) {
```

Evaluate interpolating polynomial for `y[i]` at location `x`, where `xold` $\leq$ `x` $\leq$ `xold` $+$ `h`. Note that only y is available, not y'.

```
    Doub theta=(x-xold)/h;
    Doub theta1=1.0-theta;
    Int neqn=n/2;
    if (i>=neqn) throw("no dense output for y' in StepperStoerm");
    Doub yinterp=dens[i]+theta*(dens[neqn+i]+theta1*(dens[2*neqn+i]+
        theta*(dens[3*neqn+i]+theta1*(dens[4*neqn+i]*theta+
        dens[5*neqn+i]*theta1))));
    if (mu<0)
        return yinterp;
    Doub theta05=theta-0.5;
    Doub t4=theta*theta1;
    Doub c=dens[neqn*(mu+6)+i];
    for (Int j=mu;j>0; j--)
        c=dens[neqn*(j+5)+i]+c*theta05/j;
    yinterp += t4*t4*t4*c;
    return yinterp;
}
```

```
template <class D>
void StepperStoerm<D>::dense_interp(const Int n, VecDoub_IO &y, const Int imit) {
```
Compute coefficients of the the dense interpolation formula. On input, `y[0..neqn*(imit+7)-1]` contains the dens array from `prepare_dense`. On output, these coefficients have been updated to the required values.
```
    Doub y0,y1,yp0,yp1,ypp0,ypp1,ydiff,ah,bh,ch,dh,eh,fh,gh,abh,gfh,gmf,
        ph0,ph1,ph2,ph3,ph4,ph5,fc1,fc2,fc3;
    VecDoub a(41);
    for (Int i=0; i<n; i++) {
        y0=y[i];
        y1=y[3*n+i];
        yp0=y[n+i];
        yp1=y[4*n+i];
        ypp0=y[2*n+i]/2.0;
        ypp1=y[5*n+i]/2.0;
        ydiff=y1-y0;
        ah=ydiff-yp0;
        bh=yp1-ydiff;
        ch=ah-ypp0;
        dh=bh-ah;
        eh=ypp1-bh;
        fh=dh-ch;
        gh=eh-dh;
        y[n+i]=ydiff;
        y[2*n+i]=-ah;
        y[3*n+i]=-dh;
        y[4*n+i]=gh;
        y[5*n+i]=fh;
        if (imit < 0) continue;
        abh=ah+bh;
        gfh=gh+fh;
        gmf=gh-fh;
        ph0=0.5*(y0+y1+0.25*(-abh+0.25*gfh));
        ph1=ydiff+0.25*(ah-bh+0.25*gmf);
        ph2=abh-0.5*gfh;
        ph3=6.0*(bh-ah)-3.0*gmf;
        ph4=12.0*gfh;
        ph5=120.0*gmf;
        if (imit >= 1) {
            a[1]=64.0*(y[7*n+i]-ph1);
            if (imit >= 3) {
                a[3]=64.0*(y[9*n+i]-ph3+a[1]*9.0/8.0);
                if (imit >= 5) {
                    a[5]=64.0*(y[11*n+i]-ph5+a[3]*15.0/4.0-a[1]*90.0);
                    for (Int im=7; im <=imit; im+=2) {
                        fc1=im*(im-1)*3.0/16.0;
                        fc2=fc1*(im-2)*(im-3)*4.0;
                        fc3=im*(im-1)*(im-2)*(im-3)*(im-4)*(im-5);
                        a[im]=64.0*(y[(im+6)*n+i]+fc1*a[im-2]-fc2*a[im-4]+fc3*a[im-6]);
                    }
                }
            }
        }
        a[0]=64.0*(y[6*n+i]-ph0);
        if (imit >= 2) {
            a[2]=64.0*(y[n*8+i]-ph2+a[0]*3.0/8.0);
            if (imit >= 4) {
                a[4]=64.0*(y[n*10+i]-ph4+a[2]*9.0/4.0-a[0]*18.0);
                for (Int im=6; im<=imit; im+=2) {
                    fc1=im*(im-1)*3.0/16.0;
                    fc2=fc1*(im-2)*(im-3)*4.0;
                    fc3=im*(im-1)*(im-2)*(im-3)*(im-4)*(im-5);
                    a[im]=64.0*(y[n*(im+6)+i]+a[im-2]*fc1-a[im-4]*fc2+a[im-6]*fc3);
                }
```

```
            }
        }
        for (Int im=0; im<=imit; im++)
            y[n*(im+6)+i]=a[im];
    }
}
```

CITED REFERENCES AND FURTHER READING:

Hairer, E., Nørsett, S.P., and Wanner, G. 1993, *Solving Ordinary Differential Equations I. Nonstiff Problems*, 2nd ed. (New York: Springer-Verlag). Fortran codes at www.unige.ch/~hairer/software.html.[1]