CalcLib Example of Calc::Ode

```
Calc::Ode Example
Objective: integrate acceleration of a variable mass photon rocket inline from 0 to 1
    year earth time. The photon rocket engine has a constant mass ejection rate.
    Software solves the ordinary differential equations (ode) for Special Relativity
    (SR). Compare errors with an exact solution provided by AP French.
References:
    - French, AP, Special Relativity, 1st Ed, WW Norton, 1968.
    - Goldstein, Herbert, et al, Classical Mechanics, 2nd Ed, Pearson Ed, 1992.
    - Jackson, John D, Classical Electrodynamics, 3rd Ed, Wiley, 1988.
    - Press, William H, et al, Numerical Recipes, 2nd Ed, Cambridge Press, 1992.
    - Spiegel, MR, et al, Schaum's Mathematical Handbook, 5th Ed, McGraw-Hill, 2018.
Author: Everett George Copyright (C) 2023
    https://www.jefgeorge.com
    07/04/2023
```

Output:

```
Ode Data:
   Step count (guessed) 20
   Step count (actual)  23
   End fractional mass: 0.426    Set Tolerance:       1.000 m
Photon Rocket Motor:
   Initial mass:        50000000.000 mTons 75.000% fuel
   Final fuel reserves: 8805017.000 mTons 23.480% full
   Fuel burn rate :     0.278 kg/hr
   Mass exhaust speed:  299792.469 km/s 1.000 1/c
Travel Data:
   Travel distance:     3326718246912.000 km 0.352 lightyears
   Travel ship time:    28694976.000 s  0.909 years
   Travel earth time:   31557600.000 s  1.000 years
   Time dialation:      90.929%
Final Speed Check:
   Calculated:          207657.188 km/s 0.693 1/c
   Exact:               207658.859 km/s 0.693 1/c
```

C++ Source:

```cpp
#include <iostream>

#include "Ode.h"                                        // differential equation objects

using namespace std;                                    // Standard C++ Library
using namespace calc;                                   // CalcLib namespace

enum VEC_ENUM {X=0,Y,Z,NDEM};                           // Vector 3D-Coordinates

const float VERR=0.001f;                                // integration error (km)
const float C =299792.458f;                             // speed of light (km/s)
const float C2=C*C;                                     // speed of light squared (km2/s2)

const float M0=5.0e10f;                                 // initial rocket rest mass (kg)
const float MMIN=0.25f*M0;                              // minimum payload rest mass (kg)
const float DM=-1.0e03f;                                // rest mass ejection rate (kg/s)
const float W0=C;                                       // rocket ejection speed (km/s)
const float NROC[NDEM]={-3.0f/5.0f,-4.0f/5.0f,0.0f};   // propulsion direction vector (-)

const float S2Y=365.25f*24.0f*3600.0f;                  // sec in yr
const int   NS=20;                                      // requested step count
const float T0=0.0f;                                    // start time (s)
const float TX=1.0f*S2Y;                                // end time (s)

// integration array data indexes
const int ILOC=0;                                       // location vector index
const int ISPD=ILOC+NDEM;                               // velocity vector index
const int IM=ISPD+NDEM;                                 // mass index
const int ITIM=IM+1;                                    // time index
const int NINT=ITIM+1;                                  // integration array length
float vInt[NINT];                                       // Ode integration array

void V_EQUAL(float *v1, const float *v2)                // vector equality
   {v1[X]=v2[X];
    v1[Y]=v2[Y];
    v1[Z]=v2[Z]; return;}

float V_DOT(const float *v1, const float *v2)           // vector dot product
   {float vSto=v1[X]*v2[X]+
               v1[Y]*v2[Y]+
               v1[Z]*v2[Z]; return vSto;}

float PHI_UV(const float *ur, const float *vt)          // phi factor
   {float vSto=1.0f-V_DOT(ur,vt)/C2; return vSto;}

float GAMMA2_U(const float *ur)                         // gamma^2 factor
   {float vSto=1.0f/PHI_UV(ur,ur); return vSto;}

float GAMMA_U(const float *ur)                          // gamma factor
   {float vSto=sqrt(GAMMA2_U(ur)); return vSto;}

float SIGMA_U(const float *ur)                          // sigma factor
   {float vSto=1.0f/(1.0f+sqrt(PHI_UV(ur,ur))); return vSto;}

// 2nd derivative callback function
// ALL v values represent the current state @ time t
// ALL dv values must be initialized
// All v,dv,t values evaluate in integration frame
void fnOde(float *dv, float *v, float t)                // derivative identifier function
{
       // declare variables
       float c1,c2,m0,dm,gm;
       float vFrc[NDEM],vAcc[NDEM];
       float *vSpd,*dLoc,*dVel;

       // update auxiliary derivative parameters
       m0=v[IM];                                         // get rocket mass
       vSpd=&v[ISPD]; gm=GAMMA_U(vSpd) ;                 // get rocket velocity
       dLoc=&dv[ILOC];                                   // get rocket location derivative
       dVel=&dv[ISPD];                                   // get rocket velocity derivative
       V_EQUAL(dLoc,vSpd);                               // transfer rocket velocity
       dv[IM]=DM/gm;                                     // mass derivative "gas pedal"
       dv[ITIM]=gm*PHI_UV(vSpd,vSpd);                    // determine time derivative

       // check for "out of gas" condition
       if(m0<MMIN) {
           v[IM]=MMIN;                                   // no fuel - payload only
           dv[IM]=                                       // zero mass derivative
           dVel[X]=dVel[Y]=dVel[Z]=0.0f;                 // zero acceleration
           return;                                       // coast to destination
       }

       // sum the forces in rest frame (kg/km/s^2)
       vFrc[X]=DM*W0*NROC[X];
       vFrc[Y]=DM*W0*NROC[Y];
       vFrc[Z]=DM*W0*NROC[Z];

       // determine acceleration in rest frame (F=ma)
       vAcc[X]=vFrc[X]/m0;
       vAcc[Y]=vFrc[Y]/m0;
       vAcc[Z]=vFrc[Z]/m0;

       // transform rest frame acceleration to integration frame
       c1=GAMMA2_U(vSpd);
       c2=V_DOT(vAcc,vSpd)/C2*SIGMA_U(vSpd);
       dVel[X]=(vAcc[X]-c2*vSpd[X])/c1;                   // sign of vSpd is irrelevant
       dVel[Y]=(vAcc[Y]-c2*vSpd[Y])/c1;
       dVel[Z]=(vAcc[Z]-c2*vSpd[Z])/c1;

       return;
}

// main program
int main(void)
{
       // declare input/output data
       int ns;
       float td,te,mf,rd,sp,se;
       float *vVel,*vLoc,*vFnl;

       // set initial conditions @ origin @ rest w/full gas tank
       float vRad[NDEM]={0.0f,0.0f,0.0f};               // initialize location array
       float vSpd[NDEM]={0.0f,0.0f,0.0f};               // initialize velocity array
       float vAux[NDEM]={M0,T0};                        // initialize auxiliary array

       // transfer initial conditions to integration array - vInt
       vLoc=&vInt[ILOC]; V_EQUAL(vLoc,vRad);
       vVel=&vInt[ISPD]; V_EQUAL(vVel,vSpd);
       vInt[IM]=M0;
       vInt[ITIM]=T0;

       // set up integration object
       Ode<float,float> ode(fnOde,VERR,NINT);            // initialize differential eq

       // evaluate ordinary differential equation
       try {
          vFnl=ode.eval(vInt,T0,TX,(TX-T0)/NS);          // integrate T0 to TX @ NS steps
          ns=ode.getCount(ode.CURRENT);                  // remember actual # of steps
       }
       catch(OdeErr& odeErr) {cout<<odeErr<<endl; return 1;}
       catch(...) {cout<<"Unknown execution error..."<<endl; return 1;}

       // extract data
       vLoc=&vFnl[ILOC]; {rd=sqrt(V_DOT(vLoc,vLoc));}     // fnl distance (km)
       vVel=&vFnl[ISPD]; {sp=sqrt(V_DOT(vVel,vVel));}     // fnl speed (km/s)
       mf=vFnl[IM]/M0; td=vFnl[ITIM]-T0;                  // fnl mass (kg), time (s)
       se=(1.0f-mf*mf)/(1.0f+mf*mf)*C;                    // equ rocket speed (km/s)

       // set print parameters
       cout.precision(3);                                 // set stream parameters
       cout.setf(ios::showpoint|ios::fixed|ios::right);   // set stream parameters

       // print our evaluations comparing the functions
       cout<<endl<<"   calc::Ode Class Example Application"<<endl<<endl;
       cout<<"Ode Data:"<<endl;
```

```
   cout<<"   Step count (guessed) "<<NS<<endl;
   cout<<"   Step count (actual)  "<<ns<<endl;
   cout<<"   End fractional mass: "<<mf<<endl;
   cout<<"   Set Tolerance:       "<<VERR*1000.0f<<" m"<<endl;
   cout<<"Photon Rocket Motor:"<<endl;
   cout<<"   Initial mass:        "<<M0/1000.0f<<" mTons”
       <<(1.0f-MMIN/M0)*100.0f<<"% fuel"<<endl;
   cout<<"   Final fuel reserves: "<<(mf*M0-MMIN)/1000.0f<<" mTons "
       <<mf-MMIN/M0)/(1.0f-MMIN/M0)*100.0f<<"% full"<<endl;
   cout<<"   Fuel burn rate :     "<<-DM/3600.0f<<" kg/hr"<<endl;
   cout<<"   Mass exhaust speed:  "<<W0<<" km/s "<<W0/C<<" 1/c"<<endl;
   cout<<"Travel Data:"<<endl;
   cout<<"   Travel distance:     "<<rd<<" km "<<rd/(C*S2Y)<<" lightyears"<<endl;
   cout<<"   Travel ship time:    "<<td<<" s  "<<td/S2Y<<" years"<<endl;
   cout<<"   Travel earth time:   "<<(TX-T0)<<" s  "<<(TX-T0)/S2Y<<" years"<<endl;
   cout<<"   Time dialation:      "<<(td/(TX-T0))*100.0f<<"%"<<endl;
   cout<<"Final Speed Check:"<<endl;
   cout<<"   Calculated:          "<<sp<<" km/s "<<sp/C<<" 1/c"<<endl;
   cout<<"   Exact:               "<<se<<" km/s "<<se/C<<" 1/c"<<endl;

   // clear print buffer
   cout<<endl<<flush;
   return 0;
}
```