```
#include "tridiagonal_matrix.h"
#include "twopointbvp.h"
#include "twopointbvpappr.h"
#include <iostream>
#include <fstream>

//-------------------------------------------------
// declare any needed constants
//-------------------------------------------------

double pi = acos(-1.0);

double lambda = 2.0;

//double theta = 2.3575510539e+00;

double theta = 8.5071995707e+00;

//double phi = 0.0;

int numsub = 8;

int const maxIters = 1000;
double const Toler = 1.0e-13;

//----------------------------------------------------
//set the diffusion coeffcient and if present
// the reaction, forcing function and true solution
//----------------------------------------------------

double diffusioncoeff(vector<double> &x)
{
        return 1.0;
}

double forcecoeff(vector <double> &x)
{
        return 0;
}

double reactioncoeff(vector<double> &par)
{
        return -1.0*lambda*exp(par[1]);
}

double dudr(vector<double> &par)
{
        return -1.0*lambda*exp(par[1]);
}

double truesol(vector<double> & x)
{
        double numer = cosh(theta*0.5*(x[0] - 0.5));
        double denom = cosh(0.25*theta);
        return -2*log( numer/ denom );
}

//double seed(vector<double> & par)
//{
```

```
//      return 4*phi*(par[0] - 0)*(1 - par[0]);
//}

int main()
{
        //------------------------------------------------
        //set up the two point bvp
        //------------------------------------------------
        double * dom = new double[2];
        dom[0] = 0.0;
        dom[1] = 1.0;

        TwoPointBVP *prob = new TwoPointBVP(dom, diffusioncoeff);

        double *lbval = new double[2];
        lbval[0] = 0.0; //this is gamma_0
        lbval[1] = 0.0; // this is g_0
        prob→set_left_bdry(true , lbval);

        double *rbval = new double[2];
        rbval[0] = 0.0; //this is gamma_l
        rbval[1] = 0.0;//this is g_l
        prob→set_right_bdry(true, rbval);

        prob→set_reaction(reactioncoeff,dudr);

        prob→set_forcing_function(forcecoeff);

        prob→set_true_solution(truesol);


        //-----------------------------------------------------
        //display some info about the two point bvp
        //-----------------------------------------------------
        prob→display_info_TwoPointBVP();


        //-----------------------------------------------------
        //solve for the approximate solution
        //(comment out this section if only finding error
        //higlight then ctrl + k, ctrl + c to comment out
        // highlight then ctrl+k, ctrl+u to uncomment)
        //-----------------------------------------------------

        {
                // create the 2 pt bvp approximation
                int numsubintervals = numsub;
                double * subintervals = new double[numsubintervals];
                for (int i = 0; i < numsubintervals; i++)
                {
                        subintervals[i] = (dom[1] - dom[0]) / numsubintervals;
                }

                TwoPointBVPAppr *method = new TwoPointBVPAppr(numsubintervals,
                        subintervals, prob);

                //method->set_intial_guess_seed(seed);

                vector<double> sol = method→Solve(maxIters,Toler);
                vector<double> xcoord = method→get_xcoord();

                ofstream fileout;
                fileout.open("approximatesol.txt");
                for (int i = 0; i < numsubintervals + 1; i++)
                        fileout << xcoord[i] << "\t" << sol[i] << " " << endl;
                fileout.close();

                if (prob→true_solution_is_present())
                {
                        fileout.open("truesol.txt");
                        int nres = 100;
                        double s = (dom[1] - dom[0]) / nres;
                        vector<double> x(1);
                        x[0] = dom[0];
                        for (int i = 0; i < nres + 1; i++)
                        {
                                fileout << x[0] << "\t" << prob→eval_true_soluti
on(x) << " " << endl;
                                x[0] += s;
                        }
                        fileout.close();
                }
        }

        //------------------------------------------
        //end of section that finds the approx solution
        //------------------------------------------



        //-------------------------------------------
        //find error between true and approximate soln
        //(comment out if only finding approx soln
        // higlight then ctrl+k, ctrl+c to comment out
        // highlight then ctrl+k, ctrl+u to uncomment)
        //-------------------------------------------
        {


                //{
                //      //create vectors to store hs and e(x_j)s and ln
                //      vector<double> h(10);
                //      vector<double> ln_h(10);
                //      vector<double> ex_j(10);
                //      vector<double> ln_ex_j(10);

                //      // for loop to run with diffrent size hs

                //      //counter to help update h & ex_j
                //      vector<double> doubles(10);
                //      //named doubles because each entry is a double of the pr
evious
                //      //and this vector is used to double the numsubintervals
each iteration

                //      for (int i = 0; i < 10; i++)
                //      {
                //              doubles[i] = pow(2.0, (i + 1));
                //      }

                //      //for loop to run the appx over and over w/ diff h.
                //      for (int j = 0; j < 10; j++)
                //      {

                //              // create the 2 pt bvp approximation
                //              double numsubintervals = doubles[j];
                //              double * subintervals = new double[numsubinterva
ls];
                //              for (int i = 0; i < numsubintervals; i++)
                //              {
                //                      subintervals[i] = (dom[1] - dom[0]) / nu
msubintervals;
                //              }
                //              //create subintervals size vector
                //              h[j] = (dom[1] - dom[0]) / numsubintervals;

                //              // create the approximation for the new stepinte
rval size
                //              TwoPointBVPAppr *method = new TwoPointBVPAppr(nu
msubintervals,
                //                      subintervals, prob);

                //              //method->set_intial_guess_seed(seed);

                //              //solve and   find the maximum error: e(x_j)
                //              //for the new step size h
                //              ex_j[j] = method->find_max_error(maxIters, Toler
);
                //      }

                //      //find the natural log of the subinterval lengths and er
rors
                //      //for plotting

                //      for (int i = 0; i < 10; i++)
                //      {
                //              ln_h[i] = log(h[i]);
                //              ln_ex_j[i] = log(ex_j[i]);
                //      }

                //      // output the subinterval lenght vs error to a file
                //      ofstream fileout;
                //      fileout.open("subintervallenghtvserror.txt");
                //      for (int i = 0; i < 10; i++)
                //      {
                //              fileout << h[i] << "\t" << ex_j[i] << " " << end
l;
                //      }
                //      fileout.close();

                //      // output the ln of subinterval lenght and ln of errors
to a file
                //      fileout.open("ln_subintervallenghtvsln_error.txt");
                //      for (int i = 0; i < 10; i++)
                //      {
                //              fileout << ln_h[i] << "\t" << ln_ex_j[i] << " "
<< endl;
```

```
                //      }
                //      fileout.close();

                //      //delete vars associated with the problem
                //      delete[]rbval;
                //      delete[]lbval;
                //      delete prob;
                //      delete[]dom;
                //}

        }
        //------------------------------------------------------
        // end of section that finds l infinty norm
        //-------------------------------------------------------

        return 0;
}
```