Daniel Perry
cs6210 / fall 2005
homework #4

Data Summary:

Algorithm / n	**Time (sec)**	**Iterations required**	**max|U(x,y)-u(x,y)|**	**(x[i]-x[i-1]) at termination (epsilon = 10^(-4))**
SOR				
n=10	6.99900000e-03	39	2.24926762e+00	9.53959926e-04
n=100	3.09398002e+02	243	2.71849248e+00	9.97324299e-04
n=1000				
Chebyshev				
n=10	3.57500000e-03	75	2.24933535e+00	9.71076924e-04
n=100	2.03479060e+01	479	2.72147078e+00	9.92361059e-04
n=1000	1.64032265e+02	3	1.99993214e+00	4.25803613e-05
CG				
n=10	9.08000000e-04	9	2.76352379e+00	9.65590975e-04
n=100	5.70214000e-01	6	2.03136096e+00	7.05060355e-05
n=1000	3.74071615e+02	3	2.00050875e+00	7.88013952e-04

Analysis:

The different algorithms displayed interesting pros and cons. Specifically in two main areas: speed and simplicity to implement.

Speed. Chebyshev beat the other two algorithms at two of the three sizes of n, although not because it had to do less iterations. It's speed advantage is probably due to the simplicity of the algorithm, or more due to the simplicity of calculating the q[i] and p[i] elements. Because most of the calculation for Chebyshev could be done in one loop, the speed was much faster, even though by complexity analysis it is comparable to SOR and CG (all are O(n)). SOR and CG both required multiple loops from 1 to n*n to calculate everything. However, the other two algorithms overall seemed to require **less** iterations than Chebyshev – it required twice or more as many iterations to converge. Regardless of the number of iterations required Chebyshev could still beat the other two algorithms because of only requiring one loop. One interesting characteristic displayed in CG, was that it required less and less iterations as n increased. This is probably somewhat due to the complex calculation of q[i] and p[i]. Although more complex, it clearly helps converge to the answer faster. Chebyshev, on the other hand, seemed to increase in number of iterations (except at n=1000, where it only took 3 iterations, I'm not clear as to why that happened still). SOR seemed to do the worse, as it would increase in iterations required, and didn't have the advantage of a single loop like Chebyshev.

Simplicity. As referred to above, some of the algorithms proved easier to implement than the others. Chebyshev had the simplest implementation. Again, as stated above, this is due to the simple way to compute q[i] and p[i]. They are calculated without every element of x[i]. This made it simpler to implement, as well as required only one major loop for every iteration. CG on the other hand, had a fairly complex calculation of q[i] and p[i] – it did require every element of x[i] (a dot product). This required another loop to calculate the dot product, and related quantities to get q[i].

Summarized, I think with more tuning/tweaking, CG would probably perform the best, especially as n gets larger, due to the apparent decrease in the number of iterations required as n increased. Although, Chebyshev has the simpler q[i] and p[i] elements, not requiring the dot product, so it might be better suited for a parallel implementation.

Code. The following pages include my code for implementing SOR, Chebysheve, and CG. It can also be found at http://www.cs.utah.edu/~dperry/classes/cs6210/.

Daniel Perry
cs6210 / fall 2005
homework #4

(SOR_poisson.cc)

```
/* Daniel Perry
 * cs6210 Fall 05
 * homework 3 - iterative method implementation....
 *
 * SOR on poisson equation.
 */

#include "Time.h"

#include <math.h>
#include <iostream>
using namespace std;

double poisson_matrix( int row , int col , int n );
double poisson_f( double x , double y );
double poisson_u( double x , double y );

int main(){

  int n = 100;
  int nn = n*n;
  double h = 1./(n+1);
  int col_start,col_end;
  double omega = 2. / ( 1 + sin( M_PI/(n+1) ) );
  double temp;
  double epsilon = 10e-4;
  double error, diff;

  double *U = new double[nn];
  double *X0 = new double[nn];
  double *X1 = new double[nn];
  double *X2 = new double[nn];
  double *C = new double[nn];
  double *swaper;

  Time time;
  time.initialize();
  double time_start, time_end;

  cout.setf(ios_base::scientific);
  cout.precision( 8 );

  time_start = time.currentSeconds();
  ///////////////////////////
  // Initialize X0, C, and U...
  for(int i=1; i<=n ; i++ ){
    for( int j=1; j<=n ; j++ ){
      U[(i-1)*n+(j-1)] = poisson_u( h*j , h*i );
      X0[(i-1)*n+(j-1)] = 0;// poisson_u( h*j , h*i );
      C[(i-1)*n+(j-1)] = omega * (-h*h)*poisson_f( h*j , h*i );
    }
  }

  int thing=0;
  while( thing <100000 ){

    //cout<<"X:"<<endl;

  ///////////////////////////////////
  // (1) compute the RHS vector:
  for(int i=0; i<nn ; i++ ){ //rows

    // only consider the values in the range of non-zero elements... (to improve
performance...)
    col_start = (i-n-1) > 0 ? i-n-1 : 0;
    col_end = (i+n+1) < nn ? i+n+1 : nn;

    X1[i] = 0;

    for( int j=col_start; j<col_end ; j++ ){ //cols

      // compute ((1-w)*D-w*U)*x0
      if( i==j ){ // on diagnol ( in D )
      X1[i] += (1-omega)*poisson_matrix( i , j , n ) * X0[j];
      }else if( j>i ){ // in upper (in U)
      X1[i] += (-omega)*poisson_matrix( i , j , n ) * X0[j];
      }else{
      X1[i] += 0;
      }


    }
    // now compute above+c:
    //cout<<i<<": "<<X1[i]<<endl;
    X1[i] += C[i];
  }
  //////////////////////////////////////
  // (2) solve (modified FSA)
  error = 0;
  diff = 0;
  for( int j=0 ; j<nn ; j++ ){  // loop over columns
    if( poisson_matrix(j,j,n) == 0 ){       // stop if matrix is singular
      cerr<<"*********"<<endl<<"FSA: matrix is singular, stopping"<<endl<<"(
element m["<<j<<"]["<<j<<"]=0 )"<<endl<<"************"<<endl;
      return -1;
    }
    X2[j] = X1[j] / poisson_matrix(j,j,n);  // compute solution component (note
that diagnol of poisson is the sames  as diagnole of matrix (D+wL).)
    for( int i=j+1 ; i < nn ; i++ ){  //update right-hand side.
      // this will only loop over elements in L (lower triangle): so omega*L is
poisson_matrix(i,j,n)*omega...
      X1[i] = X1[i] - (poisson_matrix(i,j,n)*omega) *X2[j];

    }

    // save the max difference between current and last iteration:
    temp = fabs( X0[j]-X2[j] );
    if( error < temp ){
      error = temp;
    }
    // save the max error between actual solution and the current solution...
    temp = fabs( X2[j]-U[j] );
    if( diff < temp ){
      diff = temp;
    }

  }

  ////////////////////////////////////////
  // (3) check if error small enough, else continue...
  if( error < epsilon && thing>2){
    //cout<<"done, err:"<<error<<endl;
    //cout<<"diff: "<<diff<<endl;
    break;
  }else{
    //cout<<"not done, err:"<<error<<endl;
    //cout<<"diff: "<<diff<<endl;
    //swap:
    swaper = X2;
    X2 = X0;
    X0 = swaper;
  }
  thing++;
  }
  time_end = time.currentSeconds();
  cout<<"n = "<< n <<endl;
  cout<<"num reps: "<<thing<<endl;
  cout<<"total seconds: "<<(time_end-time_start)<<endl;
  cout<<"err:"<<error<<endl;
  cout<<"diff: "<<diff<<endl;
  /*
  char tempchar;
  cout<<"Enter 1 to print, 0 not to print"<<endl;
  cin>>tempchar;

  if( tempchar=='1' ){
    cout<<"i: X2\t\t\tU"<<endl;
    for( int i=0; i<nn ; i++ ){
      cout<<i<<": "<<X2[i]<<"\t\t"<<U[i]<<endl;
    }
  }
  */
  return 0;
}

// defines the poisson matrix without storing it directly:
// size of thee matrix is n*n X n*n
double poisson_matrix( int row , int col , int n ){

  if( row == col ){
    return 4;
  }

  if( row==(col+1) || col==(row+1) || row==(col+n) || col==(row+n) ){
    return -1;
  }

  return 0;
}

// f(x,y) for the poisson equations:
double poisson_f( double x , double y ){
  return -2*32*(x*(1-x)+y*(1-y));
}

// u(x,y) - for the poison eqn's...
double poisson_u( double x , double y ){

  return 32*x*(1-x)*y*(1-y);

}
```

Daniel Perry
cs6210 / fall 2005
homework #4

(Chebyshev_poisson.cc)

```
/* Daniel Perry
 * cs6210 Fall 05
 * homework 3 - iterative method implementation....
 *
 * Chebyshev on poisson equation.
 */
#include "Time.h"
#include <math.h>
#include <iostream>
using namespace std;

double poisson_matrix( int row , int col , int n );
double poisson_f( double x , double y );
double poisson_u( double x , double y );

int main(){

  int n = 1000;
  int nn = n*n;
  double h = 1./(n+1);
  int col_start,col_end;
  double a = 8*sin(M_PI/(2*(n+1)))*sin(M_PI/(2*(n+1))),
    b = 8*cos(M_PI/(2*(n+1)))*cos(M_PI/(2*(n+1)));
  double temp;
  double epsilon = 10e-4;
  double error, diff;
+  double P0, Q1;

  double *U = new double[nn];
  double *X0 = new double[nn];
  double *X1 = new double[nn];
  double *X2 = new double[nn];
  double *C = new double[nn];
  double *swaper;

  Time time;
  time.initialize();
  double time_start, time_end;

  cout.setf(ios_base::scientific);
  cout.precision( 8 );

  time_start = time.currentSeconds();
  ///////////////////////////
  // Initialize X0, C, and U...
  for(int i=1; i<=n ; i++ ){
    for( int j=1; j<=n ; j++ ){
      U[(i-1)*n+(j-1)] = poisson_u( h*j , h*i );
      X0[(i-1)*n+(j-1)] = 0;// poisson_u( h*j , h*i );
      X1[(i-1)*n+(j-1)] = 0;
      C[(i-1)*n+(j-1)] = (-h*h)*poisson_f( h*j , h*i );
    }
  }
  P0 = 0;

  int thing=0;
  while( thing <100000 ){

    diff = 0;
    error = 0;
    Q1 = (b+a)/2 - P0;

    ///////////////////////////////////
    // (1) compute x[i+1] = x[i] + (1/q[i])*(p[i-1]*(x[i]-x[i-1])-r[i]); x[i+1]=X2,
x[i]=X1, x[i-1]=X0, p[i-1]=P0, q[i]=Q1;
    for(int i=0; i<nn ; i++ ){ //rows

      // only consider the values in the range of non-zero elements... (to improve
performance...)
      col_start = (i-n-1) > 0 ? i-n-1 : 0;
      col_end = (i+n+1) < nn ? i+n+1 : nn;

      temp = 0;
      for( int j=col_start; j<col_end ; j++ ){ //cols
      // compute A*X1
      temp += poisson_matrix( i , j , n ) * X1[j];
      }
      // now temp = (A*X1)[i]

      temp -= C[i];
      // now temp = r[i] = (A*X1-b)[i]

      temp = (P0*(X1[i] - X0[i]) - temp)/Q1;

      X2[i] = X1[i] + temp;

      // save the max difference between current and last iteration:
      temp = fabs( X0[i]-X2[i] );
      if( error < temp ){
      error = temp;
      }
      // save the max error between actual solution and the current solution...
      temp = fabs( X2[i]-U[i] );
      if( diff < temp ){
      diff = temp;
      }

    }

    ////////////////////////////////////////
    // (2) check if error small enough to stop, else continue...
    if( error < epsilon && thing>2){
      //cout<<"done, err:"<<error<<endl;
      //cout<<"diff: "<<diff<<endl;
      break;
    }else{
      //cout<<"not done, err:"<<error<<endl;
      //cout<<"diff: "<<diff<<endl;
      //rotate around for next iteration:
      swaper = X0;
      X0 = X1;
      X1 = X2;
      X2 = swaper;
    }


    P0 = ((b-a)/4)*((b-a)/4)/Q1;
    thing++;
  }

  time_end = time.currentSeconds();

  cout<<"result summary:"<<endl;
  cout<<"n = "<< n <<endl;
  cout<<"num reps: "<<thing<<endl;
  cout<<"total seconds: "<<(time_end-time_start)<<endl;
  cout<<"err:"<<error<<endl;
  cout<<"diff: "<<diff<<endl;

  /*
  char tempchar='0';
  cout<<"Enter 1 to print, 0 not to print"<<endl;
  cin>>tempchar;

  if( tempchar=='1' ){
    cout<<"i: X2\t\t\tU"<<endl;
    for( int i=0; i<nn ; i++ ){
      cout<<i<<": "<<X2[i]<<"\t\t"<<U[i]<<endl;
    }
  }
  */
  return 0;
}
```

```
// defines the poisson matrix without storing it directly:
// size of thee matrix is n*n X n*n
double poisson_matrix( int row , int col , int n ){

  if( row == col ){
    return 4;
  }

  if( row==(col+1) || col==(row+1) || row==(col+n) || col==(row+n) ){
    return -1;
  }

  return 0;
}

// f(x,y) for the poisson equations:
double poisson_f( double x , double y ){
  return -2*32*(x*(1-x)+y*(1-y));
}

// u(x,y) - for the poison eqn's...
double poisson_u( double x , double y ){

  return 32*x*(1-x)*y*(1-y);

}
```

Daniel Perry
cs6210 / fall 2005
homework #4

(CG_poisson.cc)
```
/* Daniel Perry
 * cs6210 Fall 05
 * homework 3 - iterative method implementation....
 *
 * CG on poisson equation.
 */
#include "Time.h"
#include <math.h>
#include <iostream>
using namespace std;

double poisson_matrix( int row , int col , int n );
double poisson_f( double x , double y );
double poisson_u( double x , double y );

int main(){

  int n = 1000;
  int nn = n*n;
  double h = 1./(n+1);
  int col_start,col_end;
  double a = 8*sin(M_PI/(2*(n+1)))*sin(M_PI/(2*(n+1))),
    b = 8*cos(M_PI/(2*(n+1)))*cos(M_PI/(2*(n+1)));
  double temp, temp2;
  double epsilon = 10e-4;
  double error, diff;
  double P0, Q1;

  double *U = new double[nn];
  double *X0 = new double[nn];
  double *X1 = new double[nn];
  double *X2 = new double[nn];
  double *C = new double[nn];
  double *swaper;

  Time time;
  time.initialize();
  double time_start, time_end;

  cout.setf(ios_base::scientific);
  cout.precision( 8 );

  time_start = time.currentSeconds();
  ///////////////////////////
  // Initialize X0, C, and U...
  for(int i=1; i<=n ; i++ ){
    for( int j=1; j<=n ; j++ ){
      U[(i-1)*n+(j-1)] = poisson_u( h*j , h*i );
      X0[(i-1)*n+(j-1)] = 0;// poisson_u( h*j , h*i );
      X1[(i-1)*n+(j-1)] = 0;
      C[(i-1)*n+(j-1)] = (-h*h)*poisson_f( h*j , h*i );
    }
  }
  P0 = 0;

  int thing=0;
  while( thing <100000 ){

    diff = 0;
    error = 0;

    ///////////////////////////////////
    // (1) compute r[i]: (storing it in X2 - it's not being used until later...)
    for(int i=0; i<nn ; i++ ){ //rows

      // only consider the values in the range of non-zero elements... (to improve
performance...)
      col_start = (i-n-1) > 0 ? i-n-1 : 0;
      col_end = (i+n+1) < nn ? i+n+1 : nn;

      X2[i] = 0;
      for( int j=col_start; j<col_end ; j++ ){ //cols
      // compute A*X1
      X2[i] += poisson_matrix( i , j , n ) * X1[j];
      }
      // now X2[i] = (A*X1)[i]

      X2[i] -= C[i];
      // now X2[i] = r[i] = (A*X1-b)[i]
    }
    ///////////////////////////////
    // (2) computer r[i]^T * A * r[i] and r[i]^T * r[i]...
    Q1 = 0;
    temp = 0;
    for( int i=0; i<nn ; i++ ){ // rows

      for( int j=col_start; j<col_end ; j++ ){ //cols
      // compute A*X1
      temp2 += poisson_matrix( i , j , n ) * X2[j];
      }
      //now temp2 = (A*r)[i]

      Q1 += X2[i]*temp2;
      temp += X2[i]*X2[i];
    }
    // now Q1 = r[i]^T * A * r[i]
    // and temp = r[i]^T * r[i]

    temp2 = temp; // same this to use at the other end of the loop (to compute P0)
    if( P0 != 0 )
      P0 *= temp2;
    // now P0 = p[i-1] = (r[i]^T * r[i])*(q[i-1]))/(r[i-1]^T * r[i-1])

    Q1 /= temp;
    Q1 -= P0;
    // now Q1 = q[i].

    ///////////////////////////////////
    // (3) compute x[i+1] = x[i] + (1/q[i])*(p[i-1]*(x[i]-x[i-1])-r[i]); x[i+1]=X2,
x[i]=X1, x[i-1]=X0, p[i-1]=P0, q[i]=Q1;
    for(int i=0; i<nn ; i++ ){ //rows

      // here X2[i] = r[i] = (A*X1-b)[i]  (computed above)
      temp = (P0*(X1[i] - X0[i]) - X2[i])/Q1;

      // now X2 = x[i+1], *not* r[i] anymore...
      X2[i] = X1[i] + temp;

      // save the max difference between current and last iteration:
      temp = fabs( X0[i]-X2[i] );
      if( error < temp ){
      error = temp;
      }
      // save the max error between actual solution and the current solution...
      temp = fabs( X2[i]-U[i] );
      if( diff < temp ){
      diff = temp;
      }

    }

    ////////////////////////////////////////
    // (2) check if error small enough to stop, else continue...
    if( error < epsilon && thing>2){
      //cout<<"done, err:"<<error<<endl;
      //cout<<"diff: "<<diff<<endl;
      break;
    }else{
      //cout<<"not done, err:"<<error<<endl;
      //cout<<"diff: "<<diff<<endl;
      //rotate around vectors for next iteration: ie x[i-1]=x[i], x[i]=x[i+1],
x[i+1]=whatever (isn't needed in next computation)
      swaper = X0;
      X0 = X1;
      X1 = X2;
      X2 = swaper;
    }
    P0 = Q1/temp2; //setting P0=p[i]=q[i]/(r[i]^T * r[i]) - which will be used in
the next iteration...(see beginning of loop).
    thing++;
  }

  time_end = time.currentSeconds();

  cout<<"result summary:"<<endl;
  cout<<"n = "<< n <<endl;
  cout<<"num reps: "<<thing<<endl;
  cout<<"total seconds: "<<(time_end-time_start)<<endl;
  cout<<"err:"<<error<<endl;
  cout<<"diff: "<<diff<<endl;
  /*
  char tempchar;
  cout<<"Enter 1 to print, 0 not to print"<<endl;
  cin>>tempchar;

  if( tempchar=='1' ){
    cout<<"i: X2\t\t\tU"<<endl;
    for( int i=0; i<nn ; i++ ){
      cout<<i<<": "<<X2[i]<<"\t\t"<<U[i]<<endl;
    }
  }
  */
  return 0;
}
// defines the poisson matrix without storing it directly:
// size of thee matrix is n*n X n*n
double poisson_matrix( int row , int col , int n ){

  if( row == col ){
    return 4;
  }

  if( row==(col+1) || col==(row+1) || row==(col+n) || col==(row+n) ){
    return -1;
  }

  return 0;
}
// f(x,y) for the poisson equations:
double poisson_f( double x , double y ){
  return -2*32*(x*(1-x)+y*(1-y));
}
// u(x,y) - for the poison eqn's...
double poisson_u( double x , double y ){
  return 32*x*(1-x)*y*(1-y);
}
```