

```

#include <stdio.h>
#define N 800

main()
{
float u[N][N],r[N][N],p[N][N],q[N][N] ;
float rdotp,rdotq,pdotq,alpha,beta;
int i,j,k;
/*****/
/* Conjugate gradient algorithm for */
/* Laplace difference equation with */
/* Dirichlet boundary conditions */
/*****/
/* Step 0: Initialize all arrays and set */
/* boundary conditions */
/*****/
for(i=0;i<N;i++)
{
for(j=0;j<N;j++)
{
u[i][j] = 0.5;
r[i][j] = 0.0;
p[i][j] = 0.0;
q[i][j] = 0.0;
}
}
for(j=0;j<N;j++) u[j][0] = 0;
for(j=0;j<N;j++) u[0][j] = 1;
for(j=0;j<N;j++) u[N-1][j] = 0;
for(j=0;j<N;j++) u[j][N-1] = 1;

/*****/
/* Step 1: Determine residual zero and */
/* search direction zero */
/*****/
for(i=1;i<N-1;i++)
{
for(j=1;j<N-1;j++)
{
r[i][j] = u[i][j-1]+u[i][j+1]+u[i-1][j]+u[i+1][j]-4.*u[i][j];    p[i][j] = r[i][j];
}
}

/* For testing purposes, do 10 CGM iterations */
for(k=1;k<=10;k++)
{

```

```

/*****
/*  Steps 2-3: Perform CG iteration    */
/*****

/*  Calculate  $q = Ap$   */

for(i=1;i<N-1;i++)
{
  for(j=1;j<N-1;j++)
  {

     $q[i][j] = 4.*p[i][j]-p[i-1][j]-p[i][j-1]-p[i+1][j]-p[i][j+1];$ 
  }
}
/*  Calculate  $r \cdot p$  and  $p \cdot q$   */

  rdotp = 0.0;
  pdotq = 0.0;
for(i=1;i<N-1;i++)
{
  for(j=1;j<N-1;j++)
  {
    rdotp = rdotp + r[i][j]*p[i][j];
    pdotq = pdotq + p[i][j]*q[i][j];
  }
}

/*  Set alpha value  */

  alpha = rdotp/pdotq;
/*  Update solution and residual  */

for(i=1;i<N-1;i++)
{
  for(j=1;j<N-1;j++)
  {
     $u[i][j] = u[i][j] + alpha*p[i][j];$ 
     $r[i][j] = r[i][j] - alpha*q[i][j];$ 
  }
}

/*  calculate beta  */

  rdotq = 0.0;

```

```
for(i=1;i<N-1;i++)
{
  for(j=1;j<N-1;j++)
  {
    rdotq = rdotq + r[i][j]*q[i][j];
  }
}
beta = rdotq/pdotq;
```

```
/* Set the new search direction */
```

```
for(i=1;i<N-1;i++)
{
  for(j=1;j<N-1;j++)
  {
    p[i][j] = r[i][j] - beta*p[i][j];
  }
}
```

```
/* A residual check and a print function needs to be added here */
```

```
} /* end for k */
```

```
} /* end main */
```