HIGH PERFORMANCE COMPUTING CENTER

HPCC

TEXAS TECH UNIVERSITY

SDE

- $Y' = f(t, Y) + g(t, Y)\ W'$ and $Y(0) = Y_0$
  where W is Brownian motion.
- Euler's method
  $Y(t+h)$ @ $Y(t) + h\ f(t, Y(t)) + h^{.5}\ g(t, Y(t))\ h$
  where h ~ N(0,1)
- Question: What is the distribution of Y(1) given Y(0)?
- Armando Arciniega and Edward Allen, Rounding error in numerical solution of stochastic differential equations, *Stochastic Analysis and Applications,* 21, 281-300 (2003).

Solving PDEs

- -Du = f, u on ¶W = g.
- Numerical approximation for W = $[0,1]^2$

U(i,j) ~ u(ih, jh) where h = 1/N.

U(i,j) =
[U(i+1,j) + U(i-1,j) + U(i,j+1) + U(i,j-1)]/4
+ h^2f(ih,jh))/4

Matrices and C

- C does not directly support numerical matrices or multidimensional arrays.

Matrices in C

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

struct matrix {
double **  mat;
int        row;
int        col;
};
```

```
struct  matrix * get_matrix(int row, int col);
struct matrix * matrix_mult(struct matrix *m1,struct matrix * m2);
void print_matrix(struct matrix *m);
double ** get_mat(int n1, int n2);

void main(){
struct   matrix  * m1,  * m2, * m3;
int i,j, n;
n = 4;
m1 = get_matrix(n,n);
m2 = get_matrix(n,n);

 printf("m1->row = %d \n", m1->row);
 printf("m1->col = %d \n", m1->col);
print_matrix(m2);
for(i=0; i<n; i++)
   for(j=0; j<n; j++){
       (m1->mat)[i][j] = 1+(1.0*i*j);
       (m2->mat)[i][j] = sin(1.0*i*j);
       }
 print_matrix(m1);
 print_matrix(m2);
 m3 = matrix_mult(m1, m2);

 print_matrix(m3);
```

```
struct  matrix * get_matrix(int row, int col)
 {struct  matrix *a;

     a = (struct matrix *) calloc(1,sizeof(struct matrix));
    if( row <1 ) {
          printf("row index less than 1\n");
          return NULL;}
    if( col <1 ) {
          printf("column index less than 1\n");
          return NULL;}
    a->row = row;
    a->col = col;
    a->mat = get_mat(row, col);
    return a;
    }
```

```
double ** get_mat(int n1, int n2)
{
int i;
   double ** mat, * temp_ptr;

         /*  Allocate space for the array  */
   temp_ptr   = (double *) calloc(n1*n2, sizeof(double));
   if((void *)temp_ptr == NULL){
     /* *inform = 4; */
      return NULL;
   }

   mat = (double **) calloc(n1, sizeof(double *));
   if((void *)(mat) == NULL){
      /**inform = 4;*/
      return NULL;
   }

   for(i=0; i< n1; i++)
      mat[i] = &(temp_ptr[i*n2]);

   return mat;
}
```

```
struct matrix * matrix_mult(struct matrix *m1, struct matrix *m2)
 {
 int i, j, k;
 struct matrix  * ans;

 ans = get_matrix(m1->row, m2->col);
for(i=0 ; i< m1->row; i++){
            for(j=0; j<m2->col; j++){
                        for(k=0; k<m1->col; k++)
                          (ans->mat)[i][j]+=((m1->mat)[i][k])*((m2-
>mat)[k][j]);}}


 return ans;
 }
```

Back to Jacobi Iteration

- $A = L + D + U$ = lower+diag+upper
- $Ax = (L+D+U)\ x = y$

- $x = -D^{(-1)}\ (L+U)x + D^{(-1)}\ y$
- Suggestes the iteration

- $x^{n+1} = -D^{(-1)}\ (L+U)x^n + D^{(-1)}\ y$

Back to our problem

- Suppose U = (Np)x(Np)
- Choose $1=k_0<k_1<k_2<\ldots<k_p=Np-1$
- Update rows [k_j, k_j -1] in processor j. Using rows [k_j-1, k_j] .
- Send answer back (gather).
- Update row k_j-1 and k_j on processor j
- Repeat until convergence!

Domain Decomposition

- Suppose $U = (Np)x(Np)$
- Choose $1=k_0<k_1<k_2<\ldots<k_p=Np-1$
- Update rows $[k_j+1,\ k_{j+1}-1]$ in processor j.
- Update rows $k_1<k_2<\ldots<k_{p-1}$
- Repeat until convergence

HIGH PERFORMANCE COMPUTING CENTER

HPCC

TEXAS TECH UNIVERSITY