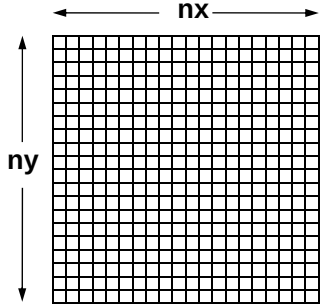
Memory Allocation
in 2-dimenstional Calculation

```
#define  NX   100  // nx = 100
#define  NY   100  // ny = 100
```

Static memory allocation:

```
double   a[NY][NX];
```

When we want to change the array size after the program runs,



Dynamic memory allocation:

1-D memory allocation is simple.

```
double   a[NX];
```

⇓

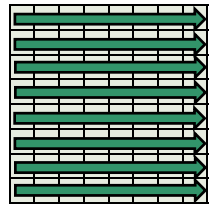
```
double   *a = (double *) malloc(nx*sizeof(double));
```


Dynamic Memory Allocation for 2-D Array

Array pointer:

```
double   **a;

a = (double **) malloc(ny*sizeof(double *));
```



TYPE-I: Source Code

```
for(i = 0; i < ny; i++) a[i] = (double *) malloc(nx*sizeof(double));
```

ny times malloc 1-D array of nx size

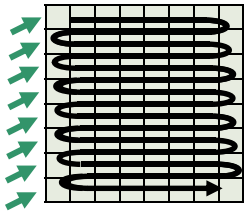
TYPE-II: Source Code

```
*a = (double *) malloc(nx*ny*sizeof(double));
```

1-D array of nx*ny size

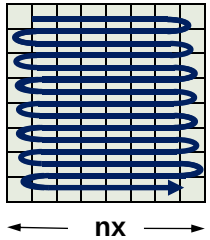
```
for(i = 0; i < ny; i++) a[i] = &((*a)[i*nx]);
```

assign pointes



Using 1-D Array as 2-D mesh

```
double   *a;
a = (double *) malloc(nx*ny*sizeof(double));
```



Self-organization:

```
                                  j = nx*jy + jx;
a[jy][jx]      →  a[j]
a[jy][jx+1]    →  a[j+1]
a[jy][jx-1]    →  a[j-1]
a[jy+1][jx]    →  a[j+nx]
a[jy-1][jx]    →  a[j-nx]
```

Using MACRO:

Source Code

```
#define   A(i, j)  A[nx*j + i]

A[nx*j + i] = (A[nx*j + i] – A[nx*j + i – 1])/dx;
```

⇒ A(i, j) = (A(i, j) – A(i-1, j)/dx;

3rd-order Upwind Scheme

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0$$

For $u \geq 0$ $\quad \frac{\partial f}{\partial x} = \frac{2f_{j+1}^n + 3f_j^n - 6f_{j-1}^n + f_{j-2}^n}{6\Delta x}$

For $u < 0$ $\quad \frac{\partial f}{\partial x} = \frac{-f_{j+2}^n + 6f_{j+1}^n - 3f_j^n - 2f_{j-1}^n}{6\Delta x}$

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = 0$$

For $u \geq 0, \quad v \geq 0$

$$\frac{\partial f}{\partial t} = -u\frac{2f_{i+1,j}^n + 3f_{i,j}^n - 6f_{i-1,j}^n + f_{i-2,j}^n}{6\Delta x} - v\frac{2f_{i,j+1}^n + 3f_{i,j}^n - 6f_{i,j-1}^n + f_{i,j-2}^n}{6\Delta y}$$

Use the Runge-Kutta 4-stage Time Integration

Accurary Check of
3rd-order Upwind Scheme

$$\frac{\partial f}{\partial t}+u\frac{\partial f}{\partial x}+v\frac{\partial f}{\partial y}=0$$

Advection velocity

$$u=1,\quad v=1$$

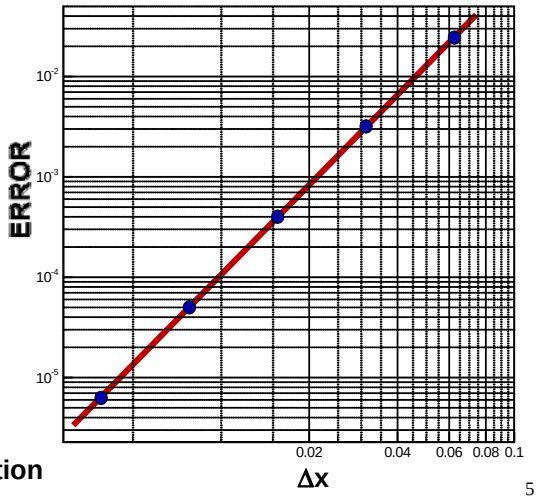
Initial Condition

$$f(x,y)=\sin(k_x x)\sin(k_y y)$$

$$\left(k_x=2\pi,\ k_y=2\pi\right)$$

Source Code

Boundary Condition:
Periodic in the x, y-direction



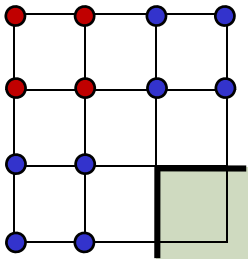
2-D Cubic Semi-Lagrangian(1/2)

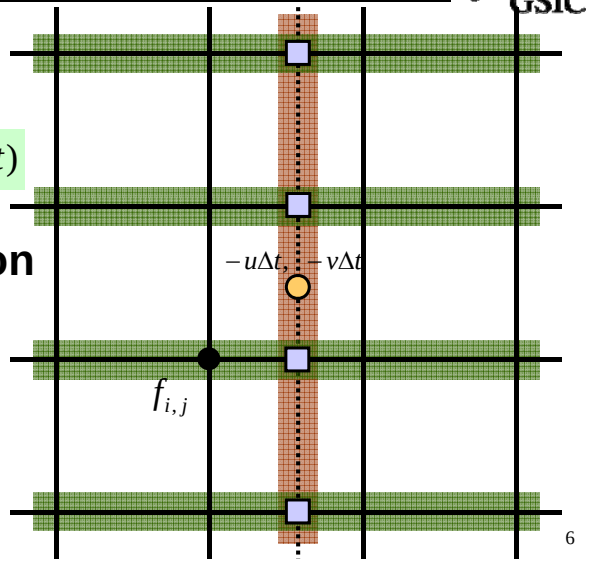
In the case

$$u_{i,j}<0,\ v_{i,j}<0$$

$$f_{i,j}^{n+1}=F_{i,j}^{n}(-u\Delta x,-v\Delta t)$$

5×1D Interpolation





2-D Cubic Semi-Lagrangian(2/2)

$$\frac{\partial f}{\partial t}+u\frac{\partial f}{\partial x}+v\frac{\partial f}{\partial y}=0$$

Advection velocity

$$u=1,\quad v=1$$

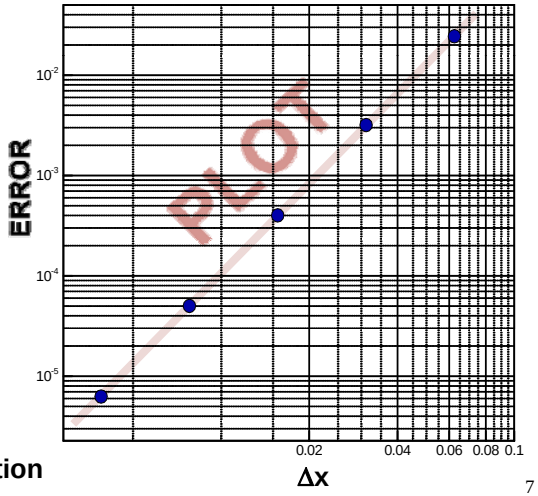
Initial Condition

$$f(x,y)=\sin(k_x x)\sin(k_y y)$$

$$\left(k_x=2\pi,\ k_y=2\pi\right)$$

Source Code

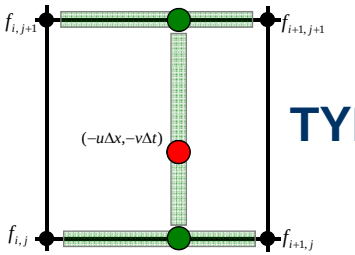
Boundary Condition:
Periodic in the x, y-direction



2-D CIP Method(1/2)

$$\frac{\partial f}{\partial t}+u\frac{\partial f}{\partial x}+v\frac{\partial f}{\partial y}=0$$

Dependent Variables: $f,\ f_x,\ f_y,\ f_{xy}$



TYPE-C

x-directional interpolation:

$$f_i^{n+1}=F(f^n,f_x^n)(-u\Delta t)$$
$$f_{x,i}^{n+1}=F_x(f^n,f_x^n)(-u\Delta t)$$
$$f_{y,i}^{n+1}=F(f_y^n,f_{xy}^n)(-u\Delta t)$$
$$f_{xy,i}^{n+1}=F_x(f_y^n,f_{xy}^n)(-u\Delta t)$$

y-directional interpolation:

$$f_i^{n+1}=G(f^n,f_y^n)(-v\Delta t)$$
$$f_{y,i}^{n+1}=G_y(f^n,f_y^n)(-v\Delta t)$$
$$f_{x,i}^{n+1}=G(f_x^n,f_{xy}^n)(-v\Delta t)$$
$$f_{xy,i}^{n+1}=G_y(f_x^n,f_{xy}^n)(-v\Delta t)$$

$$f_i^{n+1}=F(-u\Delta x)=a\xi^3+b\xi^2+f_{x,i}\xi+f_i$$

$$a=\frac{1}{\Delta x^2}\left(f_{x,i}+f_{x,i-1}\right)-\frac{2}{\Delta x}\left(f_i-f_{i-1}\right),\quad b=\frac{1}{\Delta x}\left(2f_{x,i}+f_{x,i-1}\right)-\frac{3}{\Delta x^2}\left(f_i-f_{i-1}\right)$$

2-D CIP Method(2/2)

TYPE-C

T. Aoki, J. Comp. Fluid Dynamics, 1997

$$f_j^{n+1} = F(f^n, f_x^n)(-u\Delta t) = a\xi^3 + b\xi^2 + f_{x,j}\xi + f_j$$

$$a = \frac{1}{\Delta x^2}\left(f_{x,j}^n + f_{x,j-1}^n\right) - \frac{2}{\Delta x}\left(f_j^n - f_{j-1}^n\right), \quad b = \frac{1}{\Delta x}\left(2f_{x,j}^n + f_{x,j-1}^n\right) - \frac{3}{\Delta x^2}\left(f_j^n - f_{j-1}^n\right)$$

$$f_{x,j}^{n+1} = F_x(f^n, f_x^n)(-u\Delta t) = 3a\xi^2 + 2b\xi + f_{x,j}$$

Replacing $f \to f_y \qquad f_x \to f_{xy}$

$$f_{y,j}^{n+1} = F(f_y^n, f_{xy}^n)(-v\Delta t) = c\xi^3 + d\xi^2 + f_{xy,j}^n\xi + f_{y,j}^n$$

$$c = \frac{1}{\Delta x^2}\left(f_{xy,j}^n + f_{xy,j-1}^n\right) - \frac{2}{\Delta x}\left(f_{y,j}^n - f_{y,j-1}^n\right), \quad d = \frac{1}{\Delta x}\left(2f_{xy,j}^n + f_{xy,j-1}^n\right) - \frac{3}{\Delta x^2}\left(f_{y,j}^n - f_{y,j-1}^n\right)$$

$$f_{xy,j}^{n+1} = F_x(f_y^n, f_{xy}^n)(-v\Delta t) = 3c\xi^2 + 2d\xi + f_{xy,j}^n$$

Source Code

2-D Advection-diffusion Equation

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} = \kappa\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right)$$

Advection term: 3rd-order upwind scheme:

Diffusion term: 2nd-order central difference:

For $u \geq 0, \quad v \geq 0$

$$\frac{\partial f}{\partial t} = -u\frac{2f_{i+1,j}^n + 3f_{i,j}^n - 6f_{i-1,j}^n + f_{i-2,j}^n}{6\Delta x} - v\frac{2f_{i,j+1}^n + 3f_{i,j}^n - 6f_{i,j-1}^n + f_{i,j-2}^n}{6\Delta y}$$
$$+ \kappa\left(\frac{f_{i+1,j}^n - 2f_{i,j}^n + f_{i-1,j}^n}{\Delta x^2} + \frac{f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n}{\Delta y^2}\right)$$

Use the Runge-Kutta 3-stage or 4-stage Time Integration