Quadrature Integral

Problem: Estimation of an integral with two variables Integral f(x,y) dx dy

Methodology:

Midpoint Rule

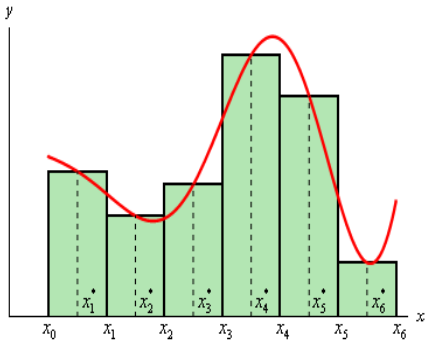
This is the rule that should be somewhat familiar to you. We will divide the interval $[a,b]$ into n subintervals of equal width,

$$\Delta x = \frac{b-a}{n}$$

We will denote each of the intervals as follows,

$$[x_0, x_1], [x_1, x_2], \ldots, [x_{n-1}, x_n] \quad \text{where } x_0 = a \text{ and } x_n = b$$

Then for each interval let x_i^* be the midpoint of the interval. We then sketch in rectangles for each subinterval with a height of $f(x_i^*)$. Here is a graph showing the set up using $n = 6$.



We can easily find the area for each of these rectangles and so for a general n we get that,

$$\int_a^b f(x)\,dx \approx \Delta x f(x_1^*) + \Delta x f(x_2^*) + \cdots + \Delta x f(x_n^*)$$

Or, upon factoring out a Δx we get the general Midpoint Rule.

$$\boxed{\int_a^b f(x)\,dx \approx \Delta x \left[f(x_1^*) + f(x_2^*) + \cdots + f(x_n^*) \right]}$$

We use the similar method to with integration estimation for single variable.

Estimate the integral of f(x,y) over [0,1]<=x<=[0,1]

Sequential Method:

```
for ( i = 1; i <= nx; i++ )
 {
  x = ( ( 2 * nx - 2 * i + 1 ) * a + ( 2 * i - 1 ) * b ) / ( 2 * nx );
  for ( j = 1; j <= ny; j++ )
  {
   y = ( ( 2 * ny - 2 * j + 1 ) * a + ( 2 * j - 1 ) * b ) / ( 2 * ny );
   total = total + f ( x, y );
  }
 }
```

OpenMP Method:

Divide the outer loop across processors.

```
# pragma omp parallel shared ( a, b, n ) private ( i, j, x, y )

# pragma omp for reduction ( + : total )

  for ( i = 1; i <= nx; i++ )
  {
    x = ( ( 2 * nx - 2 * i + 1 ) * a + ( 2 * i - 1 ) * b ) / ( 2 * nx );
    for ( j = 1; j <= ny; j++ )
    {
      y = ( ( 2 * ny - 2 * j + 1 ) * a + ( 2 * j - 1 ) * b ) / ( 2 * ny );
      total = total + f ( x, y );
    }
  }
```

MPI Version:

Brodcast all the processors about nx and ny values and find the sum parallely.

```
MPI_Bcast ( &nx, 1, MPI_INT, 0, MPI_COMM_WORLD );
MPI_Bcast ( &ny, 1, MPI_INT, 0, MPI_COMM_WORLD );

 for ( i = 1+id; i <= nx; i+=p )
  {
    x = ( ( 2 * nx - 2 * i + 1 ) * a + ( 2 * i - 1 ) * b ) / ( 2 * nx );
    for ( j = 1; j <= ny; j++ )
    {
      y = ( ( 2 * ny - 2 * j + 1 ) * a + ( 2 * j - 1 ) * b ) / ( 2 * ny );
      locTotal = locTotal + f ( x, y );
    }
  }

MPI_Reduce ( &locTotal, &total, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD );
```