CS601: Software Development for Scientific Computing

Autumn 2024

Week13 : FEM and Program Representation (Grids)

Program Representation – Structured Grids

- Grid requirements:
  - Grid dimension shall not be hardcoded
    - Consequence: implementations must define a compile-time constant
  - Grid step size shall not be hardcoded E.g. h=1/3, h=1/5 etc.
    - Consequence: can't define int arr[m][n]; //m,n to be constant expr.
  - A grid point shall be identified with cartesian coordinates / polar coordinates (e.g. with angle and radius from origin)
    - Shall be able to generate a structured grid given number of points, xi, and eta.
  - Shall allow access to any grid point
  - Shall allow for implementation of grid operators

Structured Grids - Representation

- Because of regular connectivity between cells
  - Cells can be identified with indices (x,y) or (x,y,z) and neighboring cell info can be obtained.
  - How about identifying a cell here?

    Given:

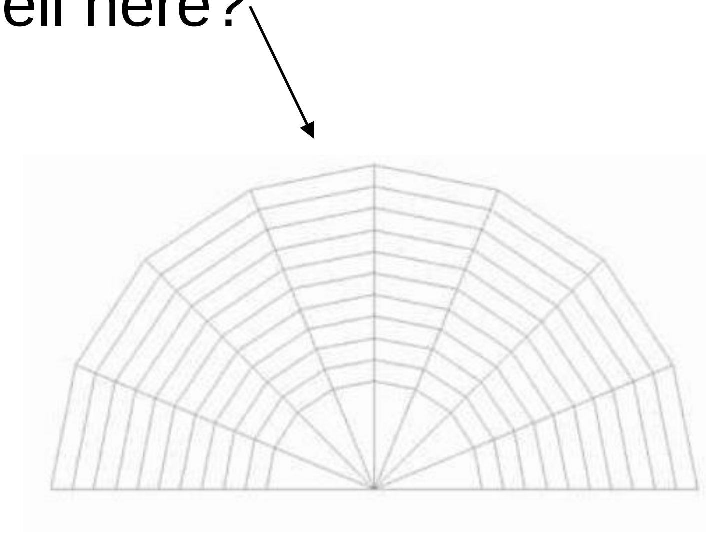
    $\xi =$ ("Xi") radius

    η = ("Eta") angle

    Compute:

    $$x = \left(\frac{1}{2} + \xi\right)\cos(\pi\eta)$$

    $$y = \left(\frac{1}{2} + \xi\right)\sin(\pi\eta)$$



class Domain

- We discretize the domain using a grid

```
class Domain{
    public:
      generate_grid(int m, int n);
      Domain(); // constructor
      //...
    private:
      //...
};
```

Method GenerateGrid

- What is the shortcoming of the following method?

```
void Domain::GenerateGrid(int m, int n) {
    if (m <=0 || n<=0)
      throw std::invalid_argument(“ERR_generate_grid”);
    else if( (xlen > 0) || (ylen > 0)) {
//there already exists a grid! Attempt to create a grid again
             delete [] x; delete [] y;
    }
    xlen=m;ylen=n; // initialize members
    x=new double[xlen*ylen]; y=new double[xlen*ylen];
}
```

- Assumes a 2D grid.

Grid Function

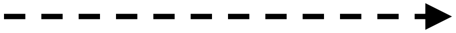
- We let a grid function to operate on the grid points
  - Example of an operator: numerical differentiation
  - Different operations possible
  - Note: grid function always operates on some grid.
  - Many functions may operate on the same grid.

```
class GridFn{
    public:
       //...
    private:
       Domain* d; //denotes aggregation relationship
       //...
};
```

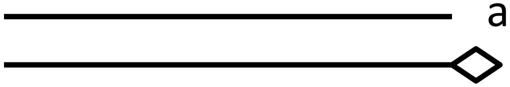
Detour: Relationships among Classes

- Dependencies ("uses")

E.g. Customer uses a MS Word editor to produce MS Word document



- Association / Aggregation ("has a")

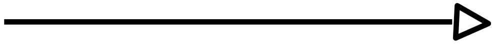


association

aggregation

E.g. Every course has a name, credits - aggregation
A student registers for course(s) – association between student and course

- Generalization ("is a")



E.g. Apple is a Fruit *(Apple and Fruit are modeled as classes, where Fruit is a super-class and Apple is a sub-class)*

Boundary conditions

- Multiple options: affect the accuracy of the solution

Name	Prescription	Interpretation
Dirichlet (essential)	u	Fixed temperature
Neumann (Natural)	$\partial u/\partial n$	Energy Flow
Robin (Mixed)	$\partial u/\partial n + f(u)$	Temperature dependent flow

- How to represent boundary conditions?
  - Create a separate Solution class

Solution

- pseudo-code

```
Domain dom; // create domain
GridFn g(dom); //create grid function to operate on a domain
Solution u(g) //prepare to compute a solution:
u.initcond() //1) set initial conditions
for(int step=0; step<maxsteps; step++) 2) iterate:
{
      u.compute(); //2) compute solution repeatedly
}
```

```
u.iterate() or u.solve()
```

class Solution

- We discretize the domain using a grid

```
class Solution{
    public:
      Solution(GridFn* d): sol(d) {}
      initcond();
      boundarycond();
      //… other member functions?
    private:
      GridFn* sol;
};
```

What is missing?

- Data array?
  - We need to make provision for storing the results of algebraic equations (temperature, displacements, stress, strain etc.)
- Type of data as template parameter?
  - Does the application accept single-precision results? Double-precision results?
- Operation on subgrids (Box)?
  - When a particular grid function is applied only in a certain region