MTH572/472:
Numerical Methods for PDES

Instructor: Alfa Heryudono

Numerically Rapid Prototyping PDEs using Mathematica

Rapid Prototyping

Using ready to use package for solving PDEs is really convenient, especially when you work with research collaborators, who concentrate more on mathematical modeling instead of numerics. Problem solving environment software such as *Mathematica* makes things easy to:

- Set partial differential equations in 1D, 2D, or even 3D.
- Set initial condition and boundary conditions.
- Choose legacy spatial discretizations: Finite-Difference, Pseudospectral, Finite-Element.
- Choose the time-stepping methods.
- Plot results.

This can be handy when you try to compare results using different methods and you simply do not have time to write those methods from scratch.

Notes:

In most basic cases, what you need are just several commands from the NDSolve framework.

Case Study I: 2D Poisson equation from Lecture 4

As an example, let us take the 2D Poisson equation from Lecture 4. For solving PDE on simple regular domain such as squares or rectangles, *Mathematica* uses finite difference by default.

Load needed package

```
Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]
```

Define the PDE operator

```
pdeop = -Laplacian[u[x, y], {x, y}]
```

```
-u^(0,2)[x, y] - u^(2,0)[x, y]
```

Define the forcing function

```
f[x_, y_] := -(4. * x^2 + y^2 - 3.) Exp[-(x^2 + 0.5 * y^2)]
```

Define the domain

```
pdedom = Rectangle[{-1, -1}, {1, 1}];
```

Define the Dirichlet boundary conditions

```
pdebcs =
  DirichletCondition[u[x, y] == Exp[-(x^2 + 0.5 * y^2)], x == 1 || x == -1 || y == -1 || y == 1];
```

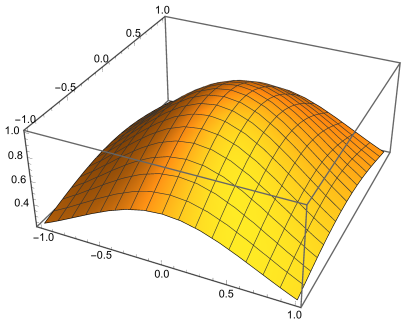
Solve the PDE

```
usol = NDSolveValue[{pdeop == f[x, y], pdebcs}, u, {x, -1, 1}, {y, -1, 1}]
```

```
InterpolatingFunction[ Domain: {{-1., 1.}, {-1., 1.}}
                       Output: scalar ]
```

Plot the solution

```
Plot3D[usol[x, y], {x, -1, 1}, {y, -1, 1}]
```



Case Study II: Dealing with irregular domain

The code can also be easily modified if you want say solving the PDE on an irregular domain and switch to finite element.

Load needed package

```
Needs["NDSolve`FEM`"]
Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]
```

Define the PDE operator

```
pdeop = -Laplacian[u[x, y], {x, y}]
```

$$-u^{(0,2)}[x, y] - u^{(2,0)}[x, y]$$

Define the forcing function

```
f[x_, y_] := -(4. * x^2 + y^2 - 3.) Exp[-(x^2 + 0.5 * y^2)]
```

Define the unit domain

```
pdedom = RegionDifference[Disk[{0, 0}, 2], Disk[{3, 0}, 2]];
```

Define the Dirichlet boundary conditions

```
pdebcs = DirichletCondition[u[x, y] == Exp[-(x^2 + 0.5 * y^2)], True];
```

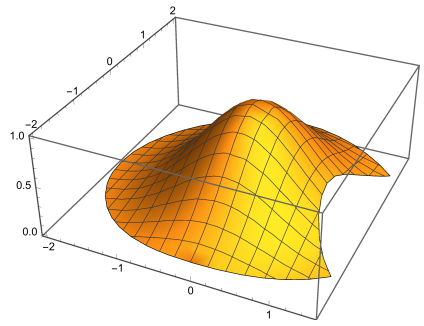
Solve the PDE

```
usol = NDSolveValue[{pdeop == f[x, y], pdebcs}, u, Element[{x, y}, pdedom]]
```

InterpolatingFunction[Domain: {{−2., 1.5}, {−2., 2.}} Output: scalar]

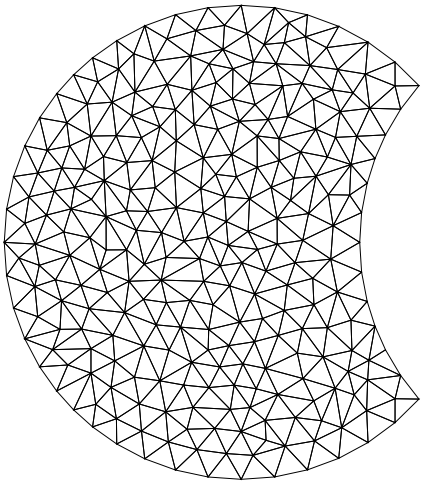
Plot the solution

```
Plot3D[usol[x, y], {x, y} ∈ pdedom]
```



Show the mesh used to find the solution

```
{mesh} = InterpolatingFunctionCoordinates[usol];
mesh["Wireframe"]
```



Case Study III: Dealing with irregular domain in 3D

You can also experiment in solving Poisson equation on a simple 3D domain and switch to finite element.

Load needed package

```
Needs["NDSolve`FEM`"]
Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]
```

Define the PDE operator

```
pdeop = -Laplacian[u[x, y, z], {x, y, z}]
```

$$-u^{(0,0,2)}[x, y, z] - u^{(0,2,0)}[x, y, z] - u^{(2,0,0)}[x, y, z]$$

Define the forcing function

```
f[x_, y_, z_] := -2.
```

Define the unit domain

```
pdedom = ImplicitRegion[x^2 + y^2 + z^2 ≤ 1, {x, y, z}];
```

Define the Dirichlet boundary conditions

```
pdebcs = DirichletCondition[u[x, y, z] == x^2 + y^2 - z^2, True];
```

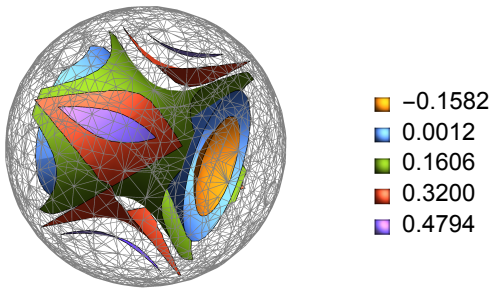
Solve the PDE

```
usol = NDSolveValue[{pdeop == f[x, y, z], pdebcs}, u, Element[{x, y, z}, pdedom]]
```

InterpolatingFunction[Domain: {{−1., 1.}, {−1., 1.}, {−1., 1.}} Output: scalar]

Show the mesh used to find the solution and plot some contours of the solutions

```
{mesh} = InterpolatingFunctionCoordinates[usol];
g1 := mesh["Wireframe"["MeshElementStyle" → EdgeForm[Gray]]]
L = 0.4 * Sqrt[2.];
g2 := ListContourPlot3D[
  Table[usol[x, y, z], {x, -L, L, 0.1}, {y, -L, L, 0.1}, {z, -L, L, 0.1}], Contours → 5,
  Mesh → None, PlotLegends → "Expressions", DataRange → {{-L, L}, {-L, L}, {-L, L}}]
Show[g1, g2]
```



Case Study IV: 2D Heat Equation from Lecture 10

You can also experiment in solving 2D Heat equation on a simple 2D domain with Method of Lines.

Load needed package

```
Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]
```

Define the PDE

```
pde = D[u[t, x, y], t] == D[u[t, x, y], x, x] + D[u[t, x, y], y, y]
```

$$u^{(1,0,0)}[t, x, y] == u^{(0,0,2)}[t, x, y] + u^{(0,2,0)}[t, x, y]$$

Define the square domain

```
pdedom = Rectangle[{-1, -1}, {1, 1}];
```

Define the Dirichlet boundary conditions

```
pdebcs = u[t, -1, y] == u[t, 1, y] == u[t, x, -1] == u[t, x, 1] == 0;
```

Define the initial condition

```
pdeic = u[0, x, y] == Sin[Pi * x] Sin[Pi * y];
```

Set solver options

```
opts = {"MethodOfLines", "SpatialDiscretization" →
    {"TensorProductGrid", "DifferenceOrder" → 4, "MaxPoints" → 50}}
```

```
{MethodOfLines,
 SpatialDiscretization → {TensorProductGrid, DifferenceOrder → 4, MaxPoints → 50}}
```

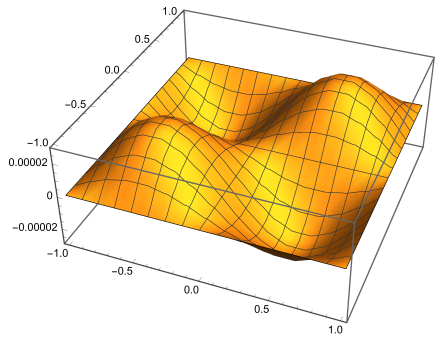
Solve the PDE

```
usol = NDSolveValue[{pde, pdeic, pdebcs},
  u, {t, 0, 0.5}, {x, -1, 1}, {y, -1, 1}, Method → opts]
```

```
InterpolatingFunction[ ⊞ Domain: {{0., 0.5}, {-1., 1.}, {-1., 1.}}
                       Output: scalar ]
```

Plot solution at t = 0.5

```
Plot3D[usol[0.5, x, y], {x, -1, 1}, {y, -1, 1}, PlotRange → All]
```



Case Study V: 2D Wave Equation on Irregular Domain

You can also experiment in solving 2D Wave equation on an irregular domain and switch to finite element.

Load needed package

```
Needs["NDSolve`FEM`"]
Needs["DifferentialEquations`InterpolatingFunctionAnatomy`"]
```

Define the PDE

```
pdeop = D[u[t, x, y], t, t] == D[u[t, x, y], x, x] + D[u[t, x, y], y, y]
```

$$u^{(2,0,0)}[t, x, y] == u^{(0,0,2)}[t, x, y] + u^{(0,2,0)}[t, x, y]$$

Define the square domain

```
pdedom = RegionDifference[Disk[], Disk[{-1 / 4, 1 / 4}, 1 / 5]];
```

Define the Dirichlet boundary conditions

```
pdebcs = DirichletCondition[u[t, x, y] == 0, True];
```

Define the initial conditions

```
pdeic = u[0, x, y] == Exp[-5 ((x - 0.2) ^ 2 + y ^ 2)];
pdedic = Derivative[1, 0, 0][u][0, x, y] == 0;
```

Solve the PDE

```
usol = NDSolveValue[{pdeop == 0, pdeic, pdedic, pdebcs},
    u, {t, 0, 2 * Pi}, Element[{x, y}, pdedom]] // Quiet;
```

Plot and animate the solution

```
frames = Table[Plot3D[usol[t, x, y], {x, y} ∈ usol["ElementMesh"], PlotRange → {-1, 1}],
   {t, 0, 2. * Pi, 2. * Pi / 20.}];
ListAnimate[frames, SaveDefinitions → True]
```