Object Oriented Programming with Applications
Lecture 9

David Šiška & Witold Gawlikowicz

School of Mathematics, University of Edinburgh

2018-19[1]

[1] Last updated 13th November 2018

Lecture 9

- C# style
- Problem Sheet 6 overview
- Example: solving 1D heat equation with finite differences

Programming style

Just like natural languages, each programming language has it's unique style. While we may be "translating" literary from a different language that we know (C++, R, Matlab, Python?) when we first learn a new language, it's important to develop a style shared by other coders using a given language (C# in our case). It leads to more coherence and optimal use of language features.

Again, this is a highly subjective and each developer/team/organization will have their own conventions, but there are some common themes which we will briefly discuss in today's lecture.

General principles

- Simplicity - Meet the expectation of your users with simple classes and simple methods.
- Clarity - Ensure that each class, interface, method variable and object has a clear purpose. Explain where, when, why and how to use each.
- Completeness - Provide the minimum functionality that any reasonable user would expect to find and use. Create complete documentation; document all features and functionality.
- Consistency - Similar entities should look and behave the same;
- Robustness - Provide predictable, documented behaviour in response to errors and expectations. Do not hide errors and do not force clients to detect errors.

General principles

- When working with an existing codebase try to adapt to it's style to assure consistency.
- Do it right the first time - aim to write clear and readable code from the start of a project.
- No standard is perfect - conventions are very useful and provide a lot of clarity, but they may not be suitable in certain situations (e.g. improving efficiency). When you deviate from a convention strive for as much clarity and consistency as possible and document why you've done so.
- Style checking tools - there are tools (e.g. FXcop, Mono.Gendarme, Resharper) that can help enforce consistent style automatically by analyzing the code and displaying warnings/errors.

Formatting - whitespaces

Use a single space to separate the keywords, parentheses, and curly braces in control flow statements:

```
for (...)
{
    //...
}
if (...)
{
    //...
}
else if (...)
{
    //...
}
else
{
    //...
}
```

Formatting - whitespaces

Use a single space on either side of a binary operators, after commas and semicolons and between parameter list in method declaration:

```
double length = Math.Sqrt(x * x + y * y);

for (int i = 0; i < 100; i++)
{
    //...
}

Vector NormalizeVector(double x, double y, double z)
{
    //...
}
```

Formatting - indentation

Use indented block statements:

```
for (int i = 0; i < N; ++i)
{
    if (...)
    {
        switch (index)
        {
            case 0:
                //..
                break;
            default:
                //...
                break;
        }
    }
```

Formatting - indentation

Break long statements into multiple lines and use indentation to group together similar elements

```
double value = Foo(Math.Pow(x, 2.0),
                   Math.Pow(y, 2.0),
                   Math.Pow(z, 2.0));

return person1.Name == person2.Name &&
       person1.Address == person2.Address &&
       person1.Phone == person2.Phone;
```

Class organization

Organise class into regions

- Public properties
- Private fields
- Constructors
- Public methods
- Private Methods

Naming

Use meaningful names, define constants

```
if (a < 65)
{
    y = 65 - a;
}
else
{
    y = 0
}
```

vs

```
if (age < RetirementAge)
{
    yearsToRetirement = RetirementAge - age;
}
else
{
    yearsToRetirement = 0
}
```

Naming

- Name according to meaning, not type - type information can be inferred from usage and context (e.g. no GetSumDouble())
- ...but suffix custom exception types with "Exception" e.g. "RootFindingException"
- Use familiar names - use words that exist in the terminology of target domain (e.g. when designing an online shop use "Customer" rather than "Client")
- Use complete words ("Message" rather than "Msg")
- Use nouns to name classes and variables and verbs to name methods
- Avoid excessively long names
- Use **PascalCase** for namespaces, classes, structures, properties, enumeration and methods
- Use **camelCase** for variables and method parameter names
- Prefix interfaces with "I" (e.g. "IComparable")

The Elements of C# Style

We've only discussed the most basic examples of C# style.

These slides have been based on the book "The Elements of C# Style" by Kenneth Baldwin, Andrewy Grey and Trevor Misfeldt.

Please refer to it for a much more detailed treatment of the subject if you're interested in more details.

Problem Sheet 6

Available at: `https://github.com/OOPA2018/Problem-sheets/blob/master/ProblemSheet6.pdf`

Example: solving 1D heat equation with finite differences

We would like to solve (approximate solution to):

$$\frac{\partial u}{\partial t} - \frac{1}{2}\frac{\partial^2 u}{\partial x^2} = 0 \text{ on } (0, T] \times (-R, R),$$

subject to $u(\cdot, -R) = u(\cdot, R) = 0$ and $u(0, \cdot) = u_0(\cdot)$ given.

Example: solving 1D heat equation with finite differences

Let

$$\delta_\tau f(t,x) := \frac{f(t,x) - f(t-\tau,x)}{\tau} \approx \frac{\partial f}{\partial t}(t,x)$$
$$\Delta_h f(t,x) := \frac{f(t,x+h) - 2f(t,x) - f(t,x-h)}{h^2} \approx \frac{\partial^2 f}{\partial x^2}(t,x).$$

Using Taylor's theorem you can check that if f is smooth and has bounded derivatives then

$$\left|\delta_\tau f(t,x) - \frac{\partial f}{\partial t}(t,x)\right| \leq c\tau$$
$$\left|\Delta_h f(t,x) - \frac{\partial^2 f}{\partial x^2}(t,x)\right| \leq ch^2,$$

with c depending on f.

Example: solving 1D heat equation with finite differences

Let $K \in \mathbb{N}$ and $J \in \mathbb{N}, J > 1$ be given and set $\tau = \frac{T}{K}$ and $h := \frac{2R}{J-1}$.

Let v be a solution to

$$\delta_\tau v - \frac{1}{2}\Delta_h v = 0 \text{ on } M_T,$$

with $v(\cdot, -R) = v(\cdot, R) = 0$ and $v(0, \cdot) = u_0(\cdot)$, where M_T is

$\{(t, x) : t = \tau, 2\tau, \ldots, T,\ x = -R+h, -R+2h, \ldots, R-2h, R-h\}.$

This v should be a good approximation of the heat equation.

Example: solving 1D heat equation with finite differences

To implement this on a computer:

Let $v^{k,j} := v(k\tau, -R + jh)$ and let us collect what we know:

1. k runs from 1 to K and j runs from 0 to $J-1$.
2. $v^{k,0} = v^{k,J-1} = 0$ for all $k = 1, \dots, K$ (boundary conditions).
3. $v^{0,j} = u_0(-R + jh)$ for all $j = 0, \dots, J-1$ (initial condition).
4. $$\frac{v^{k,j} - v^{k-1,j}}{\tau} - \frac{1}{2}\frac{v^{k,j+1} - 2v^{k,j} + v^{k,j-1}}{h^2} = 0$$
   for $k = 1, \dots, K$ and $j = 1, \dots, J-1$.

Example: solving 1D heat equation with finite differences

Consider v^k as a column vector of length J with entries $v^{k,j}$.

Then $k = 1, \dots, K$

$$\frac{v^k - v^{k-1}}{\tau} + Av^k = 0$$

if we take A to be

$$\begin{pmatrix} h^{-2} & 0 & 0 & 0 & \cdots & 0 & 0 \\ -\frac{1}{2}h^{-2} & h^{-2} & -\frac{1}{2}h^{-2} & 0 & \cdots & 0 & 0 \\ 0 & -\frac{1}{2}h^{-2} & h^{-2} & -\frac{1}{2}h^{-2} & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & -\frac{1}{2}h^{-2} & h^{-2} & -\frac{1}{2}h^{-2} & 0 \\ 0 & 0 & \cdots & 0 & -\frac{1}{2}h^{-2} & h^{-2} & -\frac{1}{2}h^{-2} \\ 0 & 0 & 0 & \cdots & 0 & 0 & h^{-2} \end{pmatrix}.$$

Example: solving 1D heat equation with finite differences

At each time step k we have to solve

$$(I - \tau A)v^k = v^{k-1}.$$

This is a linear system.

Let $S := I + \tau A$. Then we are solving $Sv^k = v^{k-1}$.

Example: solving 1D heat equation with finite differences

Our code will need:

1. Problem information: T, R, u_0.
2. Discretization information: K and J.
3. One vector to represent v^{k-1}, call it `vOld`.
4. One vector to represent v^k, call it `vNew`.
5. The sparse matricies A and S.
6. A linear solver. We will use iterative solver for sparse matricies.

Example: solving 1D heat equation with finite differences

```
class HeatEq1DFinDiffSolver
{
    // problem parameters
    private double T;
    private double R; // we are solving on (0,T) x (-R,R) with 0 bdry cond.
    private Func<double, double> initialCondition;

    // discretization parameters
    private int J; // number of steps in space
    private int K; // number of steps in time
    private double h;

    // needed for work
    private Matrix<double> A;
    private Matrix<double> S;


    // for solving linear system iteratively;
    Iterator<double> monitor;
    BiCgStab solver;
```

Example: solving 1D heat equation with finite differences

We need to construct the finite difference matrix A.

With `MathNet.Numerics` this can be done by creating a function $(i, j) \mapsto A_{ij}$. E.g. $(1, 1) \mapsto h^{-2}$.

So we write the function

```
Func<int, int, double> matrixEntry = (i, j) =>
{
    if (i == j) // diagonal entry
        return 1.0/(h*h);
    else if (i > 0 && j == i + 1)
        return -1.0 / (2 * h * h);
    else if (i < J - 1 && j == i - 1)
        return -1.0 / (2 * h * h);
    else
        return 0;
};
```

And then create `A`:

```
A = Matrix<double>.Build.Sparse(J, J, matrixEntry);
```

Example: solving 1D heat equation with finite differences

Now the matrix S:

```
double tau = T / K;
Matrix<double> I = Matrix<double>.Build.SparseIdentity(J);
S = (I - tau*A);
```

Example: solving 1D heat equation with finite differences

Iterative solver:

```
// Stop calculation if 1000 iterations reached during calculation
IterationCountStopCriterion<double> iterationCountStopCriterion
                    = new IterationCountStopCriterion<double>(100);

// Stop calculation if residuals are below 1e-8
// i.e. the calculation is considered converged
ResidualStopCriterion<double> residualStopCriterion
                    = new ResidualStopCriterion<double>(1e-8);

// Create monitor with defined stop criteria
monitor = new Iterator<double>(iterationCountStopCriterion,
                               residualStopCriterion);
solver = new BiCgStab();
```

Example: solving 1D heat equation with finite differences

We must discretize the initial condition:

```
public Vector<double> ApproxInitialCondition()
{
   Vector<double> u0 = Vector<double>.Build.Dense(J);
   for (int i = 0; i < J; ++i)
       u0[i] = initialCondition(-R + i * h);

   return u0;
}
```

Example: solving 1D heat equation with finite differences

And we solve by iterating through the time-steps:

```
public Vector<double> Solve()
{
   SetUpSolver();

   Vector<double> uOld = ApproxInitialCondition();
   Vector<double> uNew = Vector<double>.Build.Dense(J);
   for (int k = 0; k < K; k++)
   {
       // Must solve ( I - tau * A ) * uNew =  uOld i.e. S * uNew = uOld
       uNew = S.SolveIterative(uOld, solver, monitor);
       uOld = uNew;
       Console.Write("Step {0}, ", k);
   }
   Console.WriteLine();
   return uNew;
}
```