Object Oriented Programming with Applications
Lecture 7

David Šiška & Witold Gawlikowicz

School of Mathematics, University of Edinburgh

2018-19[1]

[1] Last updated 24th October 2018

Lecture 7

- Source control
- Problem Sheet 4 overview
- Numerical integration

Source control

- **Source control** (or **version control**) is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It can be used with any type of file: text-document, picture, movie, Excel spreadsheet, source code, etc.
- Most such system allow to track changes over time and across users - it's very easy to move to an earlier version of the file or check who modified a bit of code that introduced a bug to a program

Source control

Types of source control systems:

- Local source control systems
  - All changes are kept locally on a single computer
  - Possible to track changes across time, but no collaboration with additional users and everything will be lost if a computer fails
  - A simplest version of such system is saving new versions of a file with a modified name

Source control

Types of source control systems:

- Centralized source control systems - e.g. CVS, Subversion, Perforce
  - All files are kept on a single server
  - Allows collaboration with other users
  - Users check-out copies of files, modify them, and check the files back into a server where a change in the file is registered
  - Access to server required to register a change
  - While servers are usually safer than individual computers it's still a single point of failure - if a server becomes corrupt and no backups have been created all the data will be lost

Source control

Types of source control systems:

- Distributed source control systems - e.g. Git, Mercurial, Bazaar, Darcs
  - Master copy kept on a server
  - User checks out entire repository including history of all the changes
  - User can make and track multiple changes locally (Git's "commit")
  - User can get changes made by his collaborators from the server (Git's "pull")
  - User can send all his local changes to the server (Git's "push", or "sync" to "pull" and "push" with one command)
  - Even if a server fails and all the centrally stored data is lost it can be recovered using individual users' local repositories

This section is based on Git's official guide, if you want to know more visit: `https://git-scm.com/book/en/v2`

Git $\neq$ GitHub

In this course we are using **Git** and **GitHub**, these are two different (albeit related things)

- **Git** is a (distributed) source control system - it provides a way to track changes across time and users in a systematic manner by exposing relevant commands to the user ("commit", "push" etc.). It needs to be installed on a computer in order to be used. It is free and open-source.
- **GitHub** is an online hosting platform. It provides a free hosting of public (visible to everyone) git repositories. It also provides other functionality like web-interfaces, personal user pages, rendering of html websites (e.g. our syllabus), issue tracking any many others. It's been recently acquired by Microsoft. There are many alternatives, e.g.: GitLab, BitBucket, SourceForge or a private server

Using Git

There are many ways of interacting with Git repositories hosted on GitHub:

- Git command line tool
  - For many people the only "true" way of using Git
  - `https://git-scm.com/book/en/v2/Getting-Started-The-Command-Line`
- GitHub Extension for Visual Studio:
  - Allows interaction with Git repositories hosted on GitHub from within Visual Studio
  - `https://visualstudio.github.com/`
- GitHub Desktop tool:
  - Stand-alone tool allowing interaction with Git repositories hosted on GitHub via a graphical interface
  - `https://desktop.github.com/`
- GitHub website:
  - Individual files can be modified via the repository's GitHub web-page

Using Git

In this course we will use only 3 git commands:

- clone - creates a local copy of the repository, can also be achieved by downloading zip file via repository's website and extracting it locally
- commit
  - Creates a local changeset
  - Use it when you have finished making a set of related changes and the code compiles successfully
  - You need to include a comment - keep these short, but informative - e.g. "Finished implementation of Student.cs"
- sync
  - Downloads all the changes from a remote repository and uploads local changes
  - Use it often - once a few hours, at least daily to safeguard against loosing the latest changes due to hardware failure

There are many other Git commands which you are encouraged to explore by reading the official documentation.

Problem Sheet 4

Available at: `https://github.com/OOPA2018/Problem-sheets/blob/master/ProblemSheet4.pdf`

Numerical integration

Read e.g. Chapter 7 in Süli, E. and Mayers, D. - *An Introduction to Numerical Analysis*, Cambridge.

Where is this used?

- Probability: e.g. $X \sim N(0,1)$, $\mathbb{P}(X \leq 1) = ?$.
- Black–Scholes model for options with general payoff.
- CDS[2] pricing.
- And many other places in engineering, physics, etc.

[2] Credit default swap

Step 1: Newton–Cotes formulae part 1

We want to calculate

$$\int_0^1 f(x)dx.$$

The idea: we know how to integrate polynomials. Approximate f by the Lagrange polynomial p_n of order n with equally spaced interpolation points $y_i = i/n$.

Then

$$\int_0^1 f(x)dx \approx \int_0^1 p_n(x)dx.$$

Step 1: Newton–Cotes formulae part 2

Lagrange polynomials:

$$p_n(x) = \sum_{k=0}^{n} L_k(x) f(y_k) \text{ where } L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - y_i}{y_k - y_i}.$$

If we pre-calculate w_k given by

$$w_k = \int_0^1 L_k(x) dx$$

then

$$\int_0^1 f(x)dx \approx \int_0^1 p_n(x)dx = \sum_{k=0}^{n} \int_0^1 L_k(x) f(y_k) dx = \sum_{k=0}^{n} w_k f(y_k).$$

Step 1: Newton–Cotes formulae part 3

One would think that as n increases we are using higher order (hence better) approximating polynomial and so the accuracy of approximation improves.

This is only true up to a point (look up Runge phenomena).

Useful Newton–Cotes weights:

Order	w_0	w_1	w_2	w_3	w_4	Rule name
1	$1/2$	$1/2$				Trapezium
2	$1/6$	$4/6$	$1/6$			Simpson's
3	$1/8$	$3/8$	$3/8$	$1/8$		Simpson's 3/8
4	$7/90$	$32/90$	$12/90$	$32/90$	$7/90$	Boole's

Step 2: Increase accuracy - composite rules

We can increase accuracy by splitting the integration interval into N subintervals first. Let $h = (b-a)/N$ and $x_i := a + ih$. Then

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)dx$$

Approximate each integral on subinterval using Newton–Cotes of order n.

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} f(x)dx \approx \sum_{i=1}^{N} \sum_{k=0}^{n} w_k f(y_{i,k}) \qquad (1)$$

where $y_{i,k} = x_{i-1} + \frac{x_i - x_{i-1}}{n} k$.

CompositeIntegrator class

What should it do:

- Implement (1) for $n = 1, 2, 3, 4$.

What do we need for that?

- The desired order n.
- The desired N.
- The Newton cotes weights w_k.
- The quadrature points $y_{i,k}$ (store, update).
- The function values at the quadrature points $f(y_{i,k})$ (store, update).

CompositeIntegrator class: Private data

```
private int newtonCotesOrder;
const int maxOrder = 4;
const int maxOrderLength = 5;


private static double[,] weights = new double[maxOrder, maxOrderLength]
{
    {0.5, 0.5, 0, 0, 0},  // Trapezium rule
    {1.0/6.0, 4.0/6.0, 1.0/6.0, 0, 0},// Simpson's rule
    {1.0/8.0, 3.0/8.0, 3.0/8.0, 1.0/8.0,0},
    {7.0/90.0, 32.0/90.0, 12.0/90.0, 32.0/90.0, 7.0/90.0}
};


private double[] quadraturePoints;
private double[] quadraturePointsFVal;
```

CompositeIntegrator class: Constructors

```
public CompositeIntegrator()
{
    newtonCotesOrder = 1;
    quadraturePoints = new double[newtonCotesOrder+1];
    quadraturePointsFVal = new double[newtonCotesOrder+1];
}


public CompositeIntegrator(CompositeIntegrator integrator)
{
    newtonCotesOrder = integrator.newtonCotesOrder;
    quadraturePoints = new double[newtonCotesOrder+1];
    quadraturePointsFVal = new double[newtonCotesOrder+1];
}


public CompositeIntegrator(int newtonCotesOrder)
{
    this.newtonCotesOrder = newtonCotesOrder;
    quadraturePoints = new double[newtonCotesOrder+1];
    quadraturePointsFVal = new double[newtonCotesOrder+1];
}
```

CompositeIntegrator class: Private methods

```
private void UpdateQuadraturePointsAndFvals(double a, double h,
                                            Func<double, double> f)
{
    double delta = h / newtonCotesOrder;
    for (int i = 0; i <= newtonCotesOrder; i++) {
       quadraturePoints [i] = a + i * delta;
       quadraturePointsFVal[i] = f(quadraturePoints[i]);
    }
}
```

This sets up the quadrature points and values at quadrature points for the approximation of the integral

$$\int_a^{a+h} f(x)dx$$

but it doesn't do any calculations.

CompositeIntegrator class: Public method Integrate

```
public double Integrate(Func<double, double> f, double a, double b,
                        int N)
{
    double integral = 0;
    double h = (b - a) / N;
    for (int i = 0; i < N; i++)
    {
        UpdateQuadraturePointsAndFvals (a+i*h, h, f);
        double stepIncrement = 0.0;
        for (int j = 0; j <= newtonCotesOrder; j++) {
            stepIncrement += weights[newtonCotesOrder-1,j]
                             * quadraturePointsFVal [j];
        }
        integral += stepIncrement*h;
    }
    return integral;
}
```

CompositeIntegrator class: Wrapping up

We have everything: put

```
using System;
public class CompositeIntegrator
{
```

before the code from the first slide.

Indent properly.

Put

```
}
```

after the end of the Integrate method and you have the whole class.

Using CompositeIntegrator class

```
class MainClass
{
    public static void Main (string[] args)
    {
        Func<double, double> normDens
            = (x) => Math.Exp (-x * x / 2.0) / Math.Sqrt (2 * Math.PI);
        CompositeIntegrator integrator = new CompositeIntegrator (4);
        double integral = integrator.Integrate(normDens, -10, 10, 100);
    }
}
```