Object Oriented Programming with Applications
Lecture 5

David Šiška & Witold Gawlikowicz

School of Mathematics, University of Edinburgh

2018-19[1]

[1] Last updated 16th October 2018

Lecture 5

Using libraries

- Data types in `System.Collections.Generic`
  - Three examples:
    - `LinkedList`
    - `KeyValuePair`
    - `Hashtable`
- Complex numbers in `System.Numerics.Complex`.
- Linear algebra in `MathNet.Numerics.LinearAlgebra`.
  - `MathNet.Numerics.LinearAlgebra.Vector`
  - `MathNet.Numerics.LinearAlgebra.Matrix`

Read: Wright, P. - Beginning Visual C# 2005 Express Edition. Chapter 13.
Duffy, D. J. and Germani, A. - C# for Financial Markets, Chapter 5.

Key-value pairs

Used for storing key value pairs e.g. key: parameter name, value: parameter value.

Example:

```
string key = "sigma";
double value = 0.1;
KeyValuePair<string, double> keyValuePair
        = new KeyValuePair<string, double> (key, value);
```

What next? Access key and values directly via

```
keyValuePair.key;
keyValuePair.value;
```

Linked list as an alternative to arrays

Arrays can be used to store a collection of objects of the same type.

But we have to know from the beginning the length of the desired list.

If we do not then `Array.Resize` can be used to increase the array size but this is *inefficient*: it has to

- allocate a new memory block
- copy old array elements copied into the new memory
- delete the old memory (done by garbage collector).

On the other hand accessing n-th element in an array is immediate.

Linked lists

Used to store collection of objects of the same type where we don't know how many will be needed.

This is done with *references*. Each list node stores a reference to the object it has to store, to a previous item and to a next item in the list.

Adding an item to the list at the beginning or end is *efficient*: it only needs updating two references.

Getting to the n-th item in the list is inefficient it needs $O(n)$ operations.

Processing *all* the items a Linked list is as efficient as with an array.

LinkedListNode

A linked list node looks something like:

```
class LinkedListNode<T>
{
    public LinkedListNode<t> Prev, Next;
    public T value;
}
```

A linked list itself needs little data:

```
class LinkedList<T>
{
    private LinkedListNode<T> First;
}
```

Linked lists example:

Assume we have a class `OptionMarketData` which implements the `IEquatable` interface. i.e.

```
public class OptionMarketData : IEquatable<OptionMarketData>
{
    // ...
    public bool Equals(OptionMarketData other) { /* ... */ }
}
```

Can store a collection of these:

```
LinkedList<OptionMarketData> myList = new LinkedList<OptionMarketData>();
myList.AddFirst(myOptionData);
```

Linked lists example continued:

Can iterate through all items:

```
for (LinkedListNode<OptionMarketData> listNode = myList;
        listNode != null;
        listNode = listNode.Next)
{
    Console.WriteLine("Price: {0}", listNode.Value.GetPrice());
}
```

A lot to write for a common construction. C# syntax makes this easier:

```
foreach(OptionMarketData option in myList)
{
    Console.WriteLine("Price: {0}", option.GetPrice());
}
```

Costs Linked list vs. Array

Array (in C# this is resizable)

Operation	Cost	Remarks
insert	$O(n)$	will have to "shift" and resize
delete k-th entry	$O(n)$	will have to "shift"
access k-th entry	$O(1)$	`v[k]`
append two vectors	$O(m+n)$	

Linked list

Operation	Cost	Remarks
insert	$O(1)$	
delete	$O(1)$	have to find what to delete first
access k-th entry	$O(n)$	
append two lists	$O(1)$	list are simply concatented

Hashtable

Data structure for looking up "values" indexed by keys.

Example: key is student ID of type `long`, value is reference to

```
class StudentInfo
{
    private string firstName;
    private string surname;
    private string address;
    private string[] courses;
    // ... and a lot more
}
```

Set up a new hashtable

```
    Hashtable allStudents = new Hashtable();
    StudentInfo someStudent = new StudentInfo();
    // ... some code to fill in useful info about the student
    allStudents.Add(455137, someStudent);
```

Hashtable (continued)

Find if we have a student with certain ID:

```
if (!allStudents.ContainsKey(455137))
    throw new Exception('Student not found');
```

Find if we have a student with certain surname:

```
foreach (DictionaryEntry de in allStudents) {
    string surname = de.Value.GetSurname();
}
```

See also:

http://msdn.microsoft.com/en-us/library/system.collections.hashtable

Costs: Hashtable

Hashtable

Operation	Cost	Remarks
access key	average $O(1)$ worst case $O(n)$	
insert	average $O(1)$ worst case $O(n)$	
delete	average $O(1)$ worst case $O(n)$	

Other basic data structures:

We do not have time to cover all common data structures e.g.:

- Stacks,
- Queues,
- Trees,

Complex numbers

We all know that $z \in \mathbb{C}$ can be expressed as $z = a + ib$ with $a, b \in \mathbb{R}$.

So could write code that correctly handles arithmetic etc. with complex numbers.

Happily in C# this has already been done:

```
using System.Numerics;

class MainClass
{
    public static void Main (string[] args)
    {
        Complex z1 = 0.1 + 2.0 * Complex.ImaginaryOne;
        Console.WriteLine (z1);
    }
}
```

Complex numbers

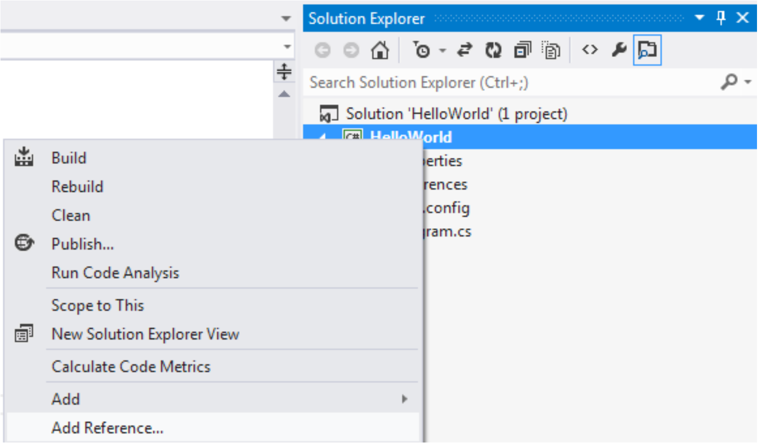
Many of the useful functions are implemented e.g. $\exp z$, $\ln z$ e.g.

```
int N = 1000;
int numRev = 4;
for (int k = 0; k < N; ++k) {
    Complex z = Complex.FromPolarCoordinates (5, 2*numRev * k * Math.PI / N);
    Console.WriteLine ("z: {0} Im(ln(z)): {1}", z, Complex.Log (z).Imaginary);
}
```

Beware of unexpected branch cuts!

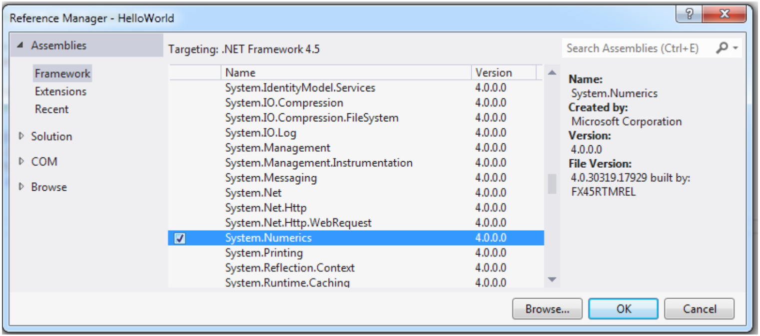
Using System.Numerics

Right click on your solution name (e.g. HelloWorld) and choose "Add Reference...".



Using System.Numerics

Find “System.Numerics” and make sure the tick-box is ticked:



A linear algebra library - MathNet

We will use a library for linear algebra called MathNet. See `http://numerics.mathdotnet.com` .

We want this mainly for linear algebra functionality.

In particular this library lets you

- Multiply vectors and matrices.
- Solve linear systems of the form $Ax = b$ (either using LU decomposition, or iteration).

Vectors and matrices: dense

Dense:

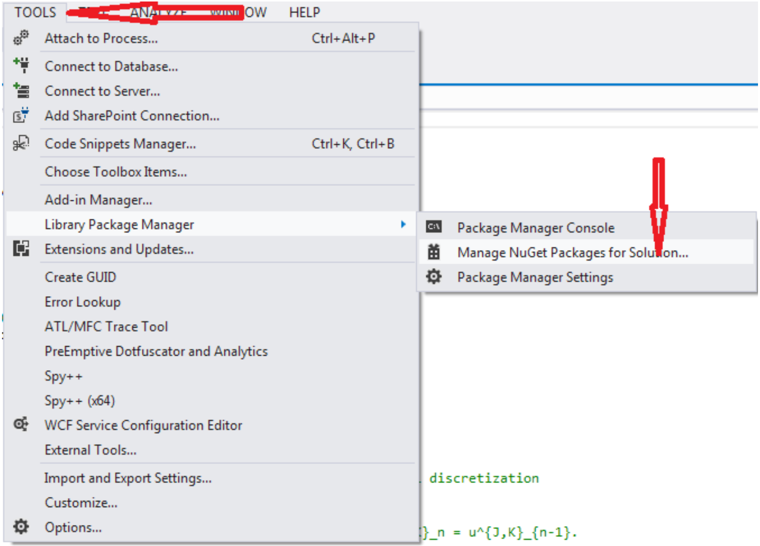
- Stored using arrays.
- Memory use for $m \times n$ double matrix is $m \times n \times$ `sizeof(double)`.
- Efficient if most entries are non-zero.
- Algorithms like LU decomposition works well for solving $Ax = b$.

Vectors and matrices: sparse

- Stored using a e.g. a dictionary of key-value pairs:
  - key is the (i, j) position in matrix,
  - value is the numerical value at the position.
- Memory use: an $m \times n$ matrix of all zeros uses no memory (the list of pairs is empty).
- Memory use: an $m \times n$ matrix with all entries non-zero uses at least $m \times n \times$ `sizeof(double)` $\times 2 \times$`sizeof(int)`.
- Efficient if most entries are zero (such matrices are common in finite-difference, finite-element methods).
- Iterative algorithms like Jacobi iterative methods work best for "solving" $Ax = b$.

Using MathNet in your own project

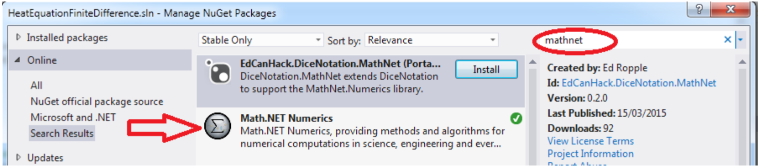
Add MathNet.Numerics to your project. Step 1



Using MathNet in your own project

Step 2:

Use the search box, find Math.NET Numerics (exactly) and press install.



Using MathNet in your own project

If the above does not work: ...

... follow instructions on the course website

MathNet and C# references - warning!

Consider

```
Vector<double> x = Vector<double>.Build.Dense(1);
Vector<double> y = x;
y[0] = 0.0;
x[0] = 10.0;
Console.WriteLine (y[0]);
```

Output?

It is 10!

MathNet and C# references - warning!

If you want a new, independent vector `y` which has the same entries as `x` initially then you need:

```
Vector<double> x = Vector<double>.Build.Dense(1);
Vector<double> y = Vector<double>.Build.DenseOfVector(x);
y[0] = 0.0;
x[0] = 10.0;
Console.WriteLine (y[0]);
```

Output?

It is 0.

Referencing other projects in the solution

- Different projects in solution don't reference each-other by default
- Assume the solution is made up of 2 projects: A and B
- If you want to expose public classes from B in A
  - In Visual Studio navigate to "Solution Explorer"
  - Right-click on project A
  - Select "Add" and "Reference..."
  - In the dialog box select "Project" in the left-hand pane
  - Tick B and click "OK"
  - Note that in "Solution Expolorer", in project A, under references, you can now see B

Summary of Lecture 5

- Data structures in `System.Collections.Generic`.
- Pairs.
- Arrays vs. lists.
- Complex numbers in `System.Numerics` and how to use it.
- Linear algebra in `MathNet.Numerics` and how to use it.