Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

MCS 471 Lecture 1
Numerical Analysis
Jan Verschelde, 22 August 2022

Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

Catalog Description

MCS 471 – Numerical Analysis

Introduction to numerical analysis; floating point arithmetic, computational linear algebra, iterative solution to nonlinear equations, interpolation, numerical integration, numerical solution of ODEs, computer subroutine packages.

Prerequisite(s): Grade of C or better in MCS 275 or grade of C or better in CS 102 or grade of C or better in CS 108; or consent of instructor.

MCS 471 is on the computational track in the MCS curriculum, next to

- MCS 320, Introduction to Symbolic Computation; and
- MCS 472, Introduction to Industrial Math & Computation.

Content and Organization of the Course

The order of topics covered in the course follows
Timothy Sauer: *Numerical Analysis*, second edition, Pearson, 2012.

1. floating-point numbers and solving equations
2. numerical methods to solve linear systems
3. interpolation, rational approximations, splines
4. data fitting, orthogonalization, least squares
5. numerical integration and differentiation (after midterm exam)
6. ordinary differential equations
7. partial differential equations
8. review for final and the final exam

Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

Purpose of the Course

The goals of MCS 471 are

1. study numerical algorithms for classical problems
2. understand difference between stability and conditioning
3. become familiar with numerical software

Software environments:

- GNU Octave
- SageMath and/or numpy, scipy
- Julia

This is a *computational* course, not a programming class.

Evaluation

- There are three exams:
  1. Two midterm exams, each worth 100 points.
  2. The final exam counts for 200 points.

  Midterm exams may be skipped.
  If skipped, then greater weight is placed on the final exam.
- Another 200 points go to five computer projects.
- Every Friday lecture ends with a quiz, except in exam weeks.
  The quizzes are worth 100 points in total.
  Homework counts as bonus. Do the homework!

The final grade is computed on the percentage of the total 700 points.

By default, unless explicitly stated that collaborations are allowed,

all submitted solutions must be your own work.

Course URL: `http://homepages.math.uic.edu/~jan/mcs471`.

Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

installing and running Julia

The current stable version of Julia is 1.7.3 (May 6, 2022).

Julia is free software. To install, do the following:

1. Visit `https://julialang.org`.
2. Click on the `Download v1.7.3` button.
3. Download and install, following the instructions.
4. Adjust the environment variable `path`.

Two ways to run Julia:

1. in an interactive Terminal or PowerShell window; or
2. at the command prompt, type

   ```
   julia program.jl
   ```

   where `program.jl` is a Julia program.

Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

the Jupyter notebook

The Jupyter notebook is a web based interface for computations.

- Cells can contain code or text.
- Your answers to homework must be in one single notebook.

To install the notebook with Julia, do the following:

1. In a Julia session, press the `]` key for the package manager.
2. Type `add IJulia`.
3. Type `using IJulia`.
4. Type `notebook()`.

The first time, the system will prompt you to install Jupyter.

Exercise 0: Install Julia and the notebook on your computer.
This is the most important exercise of this week.
Please contact me if you need help.

Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

evaluating a polynomial

A polynomial in one variable is

- a finite collection of terms,
- where each term is the product of a coefficient; and
- the variable raised to some power.

Formally, we write $p(x) = c_d x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0$, where

- c_d, c_{d-1}, ..., c_1, c_0 are the coefficients of p,
- x is the symbol for the variable of p,
- $d = \deg(p)$ is the degree of p, provided $c_d \neq 0$.

Problem Statement:

- given a polynomial and a value for its variable,
- evaluate the polynomial at the given value.

mathematical design of an algorithm

$$\begin{aligned} p(x) &= c_d x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0 \\ &= c_0 + c_1 x + \cdots + c_{d-1} x^{d-1} + c_d x^d \\ &= \sum_{i=0}^{d} c_i \cdot x^i. \end{aligned}$$

The $\sum$ is available as the `sum()` function,

a built-in function in Octave, Python, and Julia.

an array comprehension in Julia

The mathematical formula

$$p(x) = \sum_{i=0}^{d} c_i \star x^i = \sum_{i=1}^{d+1} c_i \star x^{i-1}$$

is translated in one line in Julia:

```
polyval(c, x) = sum([c[i]*x^(i-1) for i=1:length(c)])
```

which defines the function `polyval` with arguments `c` and `x`.

Julia arrays start at 1, not at 0. Note: `length(c)` $= d + 1$.

the cost of this algorithm

```
polyval(c, x) = sum([c[i]*x^(i-1) for i=1:length(c)])
```

We express the cost in terms of the formula $p(x) = \sum_{i=0}^{d} c_i \cdot x^i$

and use the degree d as the main input parameter:

- d additions in the sum $c_0 + c_1 x + \cdots + x_d x^d$,
- $d + 1$ multiplications $*$ with the coefficients, and
- $d + 1$ calls to the power function, to compute x^i.

Exercise 1: Assuming x^i costs $i - 1$ multiplications, derive the number of multiplications to evaluate a polynomial of degree d by the algorithm as defined by `polyval` above.

accumulating the powers of the value for x

```
"""
Returns the value of the polynomial with
coeffients in cff at the argument arg.
The coefficient of the constant term is at cff[1].
The powers of the argument are accumulated.
"""
function evaluate(cff::Array, arg::Float64)
    result = cff[1] # constant coefficient
    argpow = 1.0    # accumulates power of argument
    for i = 2:length(cff)
        argpow = argpow*arg # arg^(i-1)
        result = result + cff[i]*argpow
    end
    return result
end
```

Observe the $*$ in the `for` loop $\Rightarrow$ twice as many $*$ as $+$.

nesting multiplication with addition

Rewrite a quadric:
$c_2 \star x^2 + c_1 \star x + c_0 = (c_2 \star x + c_1) \star x + c_0$.

Rewrite a cubic:
$c_3 \star x^3 + c_2 \star x^2 + c_1 \star x + c_0 = ((c_3 \star x + c_2) \star x + c_1) \star x + c_0$.

Rewrite a quartic:
$$c_4 \star x^4 + c_3 \star x^3 + c_2 \star x^2 + c_1 \star x + c_0 = (((c_4 \star x + c_3) \star x + c_2) \star x + c_1) \star x + c_0.$$

We count as many multiplications $\star$ as additions $+$.

Exercise 2: Show (proof by induction) that evaluation of a polynomial of degree d can be done with d multiplications and d additions.

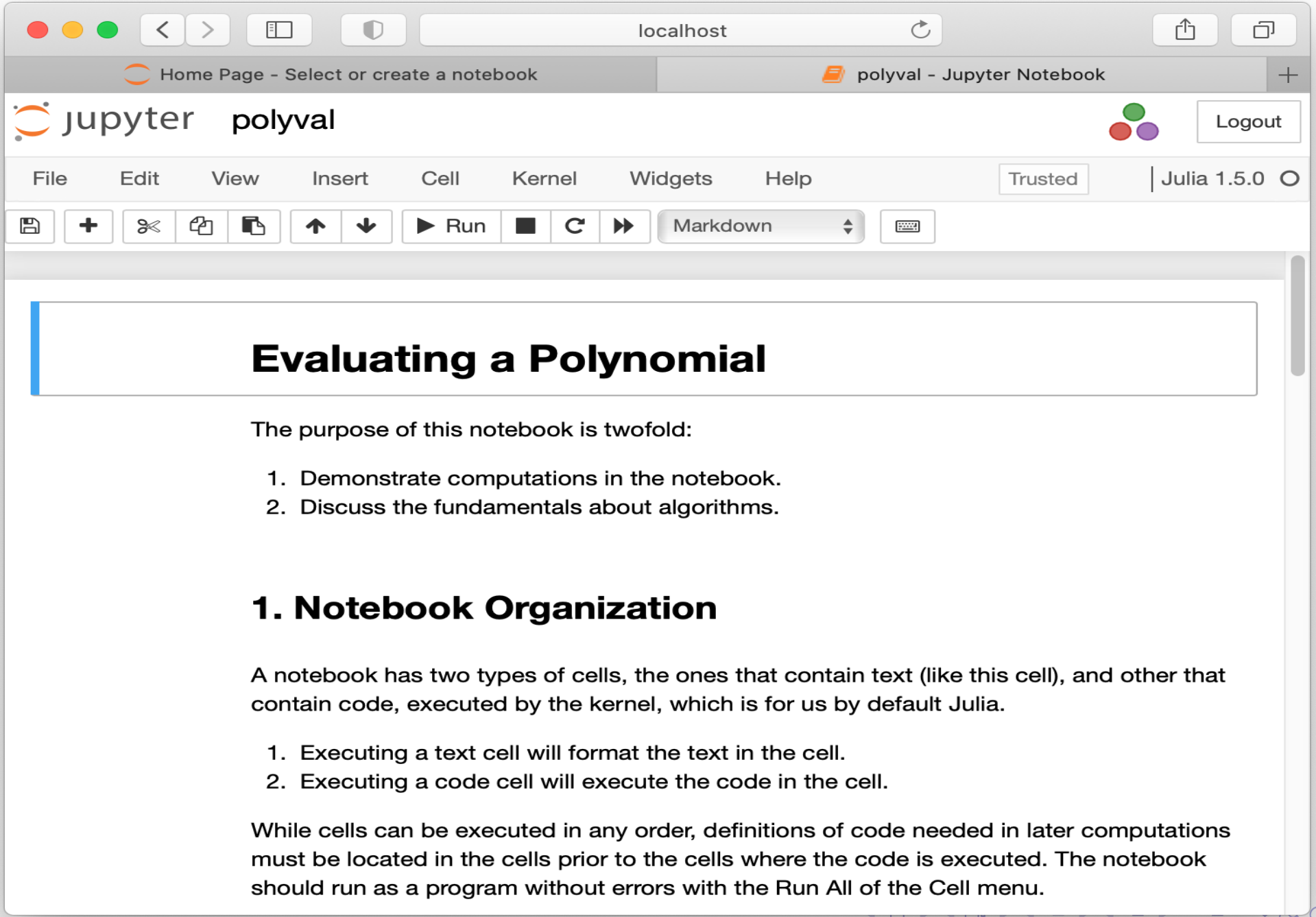
Horner's method

```
"""
Returns the value of the polynomial with
coeffients in cff at the argument arg.
The coefficient of the constant term is at cff[1].
Applies the nested scheme of Horner.
"""
function Horner(cff::Array, arg::Float64)
    index = length(cff)
    result = cff[index]
    index = index - 1
    while index > 0
        result = result*arg + cff[index]
        index = index - 1
    end
    return result
end
```

We see as many floating-point multiplications as additions.

the function `main()` in the script `polyval.jl`

```
"""
Prompts the user for the degree
and generates the coefficients
of a polynomial in one variable.
"""
function main()
    print("Give the degree : ")
    line = readline(stdin)
    deg = parse(Int, line)
    cff = rand(deg+1)
    strcff = string(cff)
    println("coefficients : $strcff")
    print("Give the value for x : ")
    line = readline(stdin)
    arg = parse(Float64, line)
    strarg = string(arg)
    val = polyval(cff, arg)
    strval = string(val)
    println("The value at $strarg : $strval")
    ...
```


localhost
Home Page - Select or create a notebook
polyval - Jupyter Notebook
jupyter polyval
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.5.0
Run
Markdown
Evaluating a Polynomial
The purpose of this notebook is twofold:
1. Demonstrate computations in the notebook.
2. Discuss the fundamentals about algorithms.
1. Notebook Organization
A notebook has two types of cells, the ones that contain text (like this cell), and other that contain code, executed by the kernel, which is for us by default Julia.
1. Executing a text cell will format the text in the cell.
2. Executing a code cell will execute the code in the cell.
While cells can be executed in any order, definitions of code needed in later computations must be located in the cells prior to the cells where the code is executed. The notebook should run as a program without errors with the Run All of the Cell menu.

Welcome to MCS 471

1. About the Course
   - content
   - expectations
2. Julia
   - installing and running
   - the Jupyter notebook
3. Fundamentals
   - evaluating a polynomial
   - binary numbers

binary numbers

A binary number is represented as a sequence of bits.
A bit is either 0 or 1. For example: `101101.010001`.

The value of the integer part of `101101.010001` is

$$\begin{aligned} \texttt{101101} &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 32 + 8 + 4 + 1 = 45. \end{aligned}$$

The value of the fraction part of `101101.010001` is `.010001` =

$$0 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^2 + 0 \cdot \left(\frac{1}{2}\right)^3 + 0 \cdot \left(\frac{1}{2}\right)^4 + 0 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^6$$
$$= \frac{1}{4} + \frac{1}{64} = \frac{17}{64}.$$

running a Julia script

```
$ julia binarynum.jl
Give the number of bits in the integer part : 6
Bits in a random integer : [1, 0, 1, 1, 0, 1].
A random integer in binary : 101101.
Give the number of bits in the integer part : 6
Bits in a random integer : [0, 1, 0, 0, 0, 1].
A random integer in binary : 010001.
A random binary number : 101101.010001.
The value of the integer part 101101 : 45.
The value of the fraction part 010001 : 17//64.
The value of 101101.010001 : 2897//64.
```

binary expansion of an integer

The remainder of division by 2 gives the least significant bit.

```
"""
Returns the array of bits in the binary
expansion of the given integer number n.
"""
function tobinary(n::Int64)
    bits = [] # array of anything
    nbr = n
    while nbr > 0
        rmd = rem(nbr, 2)   # remainder
        nbr = div(nbr, 2)   # quotient
        push!(bits, rmd)    # append bit
    end
    # revert the order of the bits
    result = [bits[k] for k=length(bits):-1:1]
    return result
end
```

converting a fraction into binary

$$\frac{17}{64} = \frac{16}{64} + \frac{1}{64} = \frac{1}{4} + \frac{1}{64} = 1 \cdot \left(\frac{1}{2}\right)^2 + 1 \cdot \left(\frac{1}{2}\right)^6 = 0.010001$$

We can find the powers of $\frac{1}{2}$ by multiplication by 2:

$$\begin{aligned}
\frac{17}{64} \times 2 &= \frac{34}{64} + 0 \\
\frac{34}{64} \times 2 &= \frac{4}{64} + 1 \\
\frac{4}{64} \times 2 &= \frac{8}{64} + 0 \\
\frac{8}{64} \times 2 &= \frac{16}{64} + 0 \\
\frac{16}{64} \times 2 &= \frac{32}{64} + 0 \\
\frac{32}{64} \times 2 &= \frac{64}{64} = 1
\end{aligned}$$

binary expansion of 1/10 = 0.000110011...

$$
\begin{aligned}
\frac{1}{10} \times 2 &= \frac{2}{10} + 0 \\
\frac{2}{10} \times 2 &= \frac{4}{10} + 0 \\
\frac{4}{10} \times 2 &= \frac{8}{10} + 0 \\
\frac{8}{10} \times 2 &= \frac{6}{10} + 1 \\
\frac{6}{10} \times 2 &= \frac{2}{10} + 1 \\
\frac{2}{10} \times 2 &= \frac{4}{10} + 0 \\
\frac{4}{10} \times 2 &= \frac{8}{10} + 0 \\
\frac{8}{10} \times 2 &= \frac{6}{10} + 1 \\
\frac{6}{10} \times 2 &= \frac{2}{10} + 1 \\
&\vdots
\end{aligned}
$$

tell the story of binary number expansions

Download `polyval.ipynb` and `binarynum.jl`.

1. `polyval.ipynb` is the notebook version of `polyval.jl`.
2. `binarynum.jl` explores binary number expansions.

Exercise 3: Make a Jupyter notebook to document and execute the computations in `binarynum.jl`.

1. Explain the computations *in your own words*.
2. Do not forget the punch line:

$$\frac{1}{10} \neq 0.1.$$

recommended reading

- Julia Tutorials at `https://julialang.org/learning`.
- Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah: **Julia: A Fresh Approach to Numerical Computing.** *SIAM Review*, volume 59, no 1, pages 65–98, 2017.
- Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman, et al.: **Julia Language Documentation.**

`https://media.readthedocs.org/pdf/julia/latest/julia.pdf`

- **Introducing Julia**

  `https://en.wikibooks.org/wiki/Introducing_Julia`