Лабораторная работа

Кириллов Р.А. @ КН-402

18 октября 2013 г.

Функция: $f(x) = \dfrac{5}{1 + 5e^{-0.5x}}$

$$f'(x) = \frac{12.5e^{0.5x}}{(5 + e^{0.5x})^2}$$

$$f''(x) = \frac{e^{-0.5x} \cdot (31.25e^{x} - 6.25e^{1.5x})}{(5 + e^{0.5x})^3}$$

Полином H_1:

Узлы:
$x_0 = 0$, кратность 3; значения функции и производных:
$f(x_0) = 0.8333333333333334$
$f(x_0)' = 0.3472222222222222$
$f(x_0)'' = 0.11574074074074074$

$x_1 = 5$, кратность 1; значения функции и производных:
$f(x_1) = 3.5450307704355817$

$x_2 = 10$, кратность 1; значения функции и производных:
$f(x_2) = 4.837041358471184$

$$H_1 = f(x_0) +$$
$$f(x_0, x_0) \cdot (x - x_0)^1 +$$
$$f(x_0, x_0, x_0) \cdot (x - x_0)^2 +$$
$$f(x_0, x_0, x_0, x_1) \cdot (x - x_0)^3 +$$
$$f(x_0, x_0, x_0, x_1, x_2) \cdot (x - x_0)^3 \cdot (x - x_1)^1$$

$$f(x_0) = 0.8333333333333334$$
$$f(x_0, x_0) = 0.3472222222222222$$
$$f(x_0, x_0, x_0) = 0.11574074074074074$$
$$f(x_0, x_0, x_0, x_1) = -0.01534345754021905$$
$$f(x_0, x_0, x_0, x_1, x_2) = 0.0008601738538121208$$

Полином H_2:

Узлы:
$x_0 = 0$, кратность 1; значения функции и производных:
$f(x_0) = 0.8333333333333334$

$x_1 = 5$, кратность 2; значения функции и производных:
$f(x_1) = 3.5450307704355817$
$f(x_1)' = 0.5157910688842814$

$x_2 = 10$, кратность 1; значения функции и производных:
$f(x_2) = 4.837041358471184$

$$H_2 = f(x_0) +$$
$$f(x_0, x_1) \cdot (x - x_0)^1 +$$
$$f(x_0, x_1, x_1) \cdot (x - x_0)^1 \cdot (x - x_1)^1 +$$
$$f(x_0, x_1, x_1, x_2) \cdot (x - x_0)^1 \cdot (x - x_1)^2$$

$$f(x_0) = 0.8333333333333334$$
$$f(x_0, x_1) = 0.5423394874204497$$
$$f(x_0, x_1, x_1) = -0.005309683707233659$$
$$f(x_0, x_1, x_1, x_2) = -0.004616810654819851$$

Полином H_3:

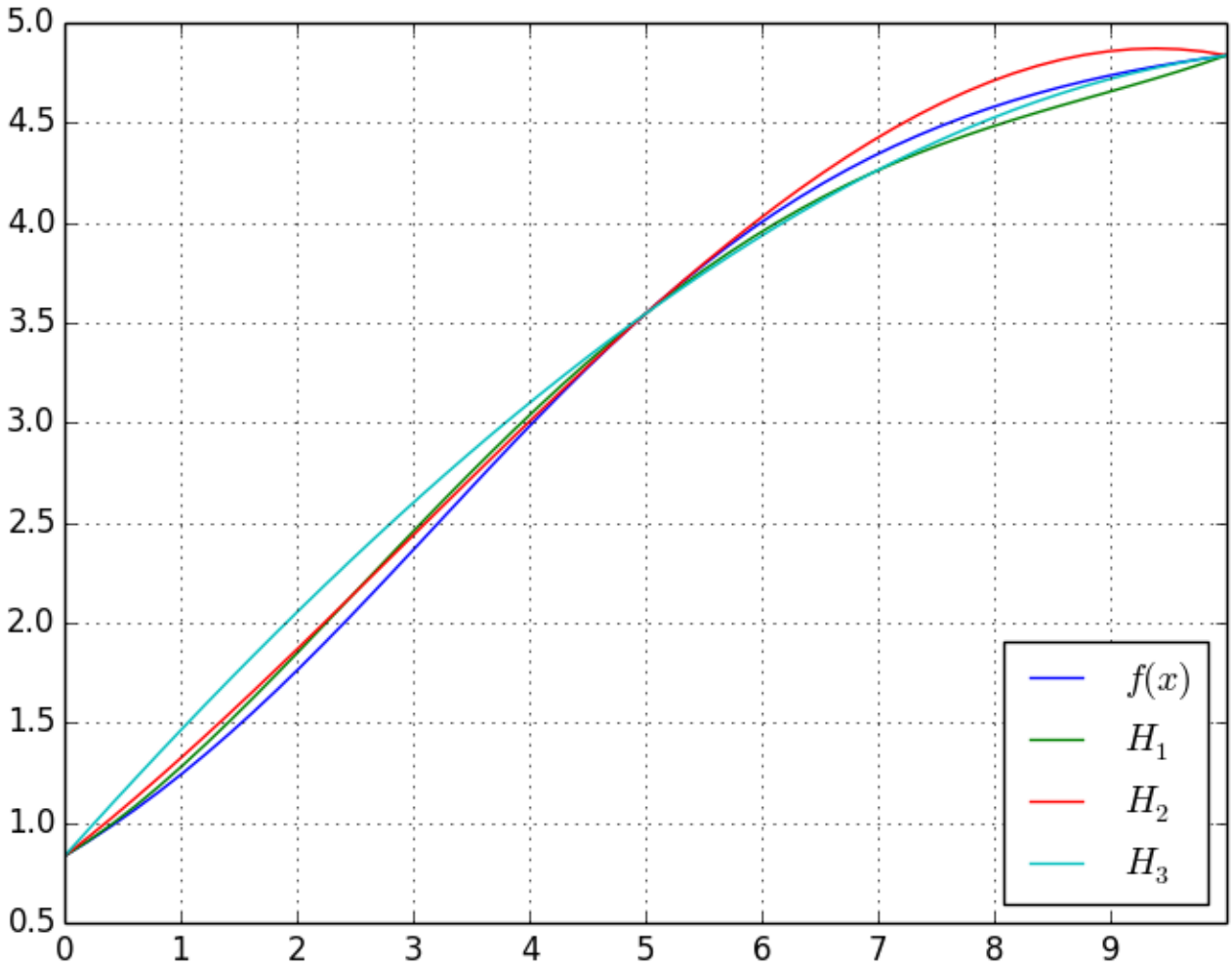
Узлы:
$x_0 = 0$, кратность 1; значения функции и производных:
$f(x_0) = 0.8333333333333334$

$x_1 = 5$, кратность 1; значения функции и производных:
$f(x_1) = 3.5450307704355817$

$x_2 = 10$, кратность 2; значения функции и производных:
$f(x_2) = 4.837041358471184$
$f(x_2)' = 0.07882376887951631$

$$H_3 = f(x_0) +$$
$$f(x_0, x_1) \cdot (x - x_0)^1 +$$
$$f(x_0, x_1, x_2) \cdot (x - x_0)^1 \cdot (x - x_1)^1 +$$
$$f(x_0, x_1, x_2, x_2) \cdot (x - x_0)^1 \cdot (x - x_1)^1 \cdot (x - x_2)^1$$

$$f(x_0) = 0.8333333333333334$$
$$f(x_0, x_1) = 0.5423394874204497$$
$$f(x_0, x_1, x_2) = -0.028393736981332914$$
$$f(x_0, x_1, x_2, x_2) = -0.0007521932764187937$$



Код на python, использованный для вычислений: (`github.com/Rast1234/hermit`)

```
# coding=utf-8

__author__ = 'rast'

from math import e
import matplotlib.pyplot as plt
import matplotlib
import numpy
```

```
from functools import reduce
from operator import mul
from sys import argv

def f(x, dx=0):
    """
    Function and diff values
    """
    functions = {
        0: lambda x: 5/(1+5*e**(-0.5*x)),
        1: lambda x: (12.5*e**(0.5*x))/(5+e**(0.5*x))**2,
        2: lambda x: (e**(-0.5*x)*(31.25*e**x-6.25*e**(1.5*x)))/(5+e**(0.5*x)
    }
    return functions[dx](x)

class Node(object):
    """
    Named node with specified multiplicity.
    Stores function values calculated once.
    """

    def __init__(self, point, multiplicity):
        self.x = point[1]
        self.multiplicity = multiplicity
        self.values = []
        self.name = point[0]
        for i in range(self.multiplicity):
            self.values.append(f(self.x,i))

    def __repr__(self):
        return self.name
    def printTex(self):
        return "${0} = {1}$, кратность ${2}$; значенияфункцииипроизводных
: ${3}$".format(
            self.name,
            self.x,
            self.multiplicity,
            r"\\" + r"\\".join(["f({{{0}}})^{{{1}}} = {{{2}}}".format(self.na
        )

class DivDifference(object):
    """Pretty representation of divided difference
    f(x0)(x-x0)^0
    f(x0, x0)(x-x0)^1
    f(x0, x0, x0)(x-x0)^2
```

```
f(x0, x0, x0, x1)(x-x0)^3(x-x1)^0
"""
def __init__(self, repeatedNodes):
    """Takes list of (node, repeat)
    """
    self.repeatedNodes = repeatedNodes

def calculate(self):
    """
    Recursive calculations
    """
    if len(self.repeatedNodes) == 1:
        (node, power) = self.repeatedNodes[0]
        #print("power=",power)
        #print("values=",node.values)
        power -= 1
        value = node.values[power]
        #print("value=", value)
        return value
    else:
        withoutFirst = DivDifference(self.removeFirst())
        withoutLast = DivDifference(self.removeLast())
        first = self.repeatedNodes[0][0].x
        last = self.repeatedNodes[-1][0].x
        return (withoutLast.calculate() - withoutFirst.calculate())/(firs

def removeFirst(self):
    """
    x0,x0,x0,x1 -> x0,x0,x1
    x0,x1 -> x1
    """
    (node, power) = self.repeatedNodes[0]
    if power != 1:
        return [(node, power-1)] + self.repeatedNodes[1:]
    else:
        return self.repeatedNodes[1:]

def removeLast(self):
    """
    x0,x0,x0,x1,x1 -> x0,x0,x0,x1
    x0,x1 -> x0
    """
    (node, power) = self.repeatedNodes[-1]
    if power != 1:
        return self.repeatedNodes[:-1] + [(node, power-1)]
```

```
        else:
            return self.repeatedNodes[:-1]

    def f(self, x):
        """
        Expected polynomial part behavior
        """
        seq = []
        for (node, repeat) in self.repeatedNodes:
            for i in range(repeat):
                seq.append(node)

        stuff = []  # will contain (node, power) like (x_0, 2)
        prev = seq[0]  # assume there are more than zero elements
        power = 0
        for node in seq[1:]:
            if node == prev:  # grow power!
                power += 1
            else:  # finalize
                stuff.append((prev, power+1))
                power = 0
            prev = node
        # don't forget the last one!
        stuff.append((seq[-1], power))
        print(stuff)
        return self.calculate() * reduce(mul, ([self.__expression(x,data) for

    def __expression(self, x, nodeAndPower):
        """
        calculate (x-x_n)^m
        """
        nodeValue = nodeAndPower[0].x
        power = nodeAndPower[1]
        return (x - nodeValue)**power

    def printFactor(self):
        """f(...)
        """
        seq = self.__generateSequence()
        return "f({})".format(", ".join(seq))

    def printExpression(self):
        """(x-...)^2 * (x-...)^1 * ...

        a => 0
```

```
        a,a => 1
        a,a,b => 2,0
        a,a,b,b => 2,1
        a,a,b,b,b => 2,2
        a,a,b,b,b,c => 2,3,0дляпоследовательного

          a, степень = n-1когдапоявляетсяновыйэлемент
            (b), степеньдля  a = n

        """
        seq = self.__generateSequence()
        stuff = []  # will contain (name, power) like (x_0, 2)
        prev = seq[0]  # assume there are more than zero elements
        power = 0

        for x in seq[1:]:
            if x == prev:  # grow power!
                power += 1
            else:  # finalize
                stuff.append((prev, power+1))
                power = 0
            prev = x
        # don't forget the last one!
        stuff.append((seq[-1], power))

        exprs = ["(x - {0})^{1}".format(x, power) if power != 0 else "" for (
        return "\cdot".join(filter(None, exprs))

    def printTex(self):
        """print valid TeX representation
        """
        left = self.printFactor()
        right = self.printExpression()
        return "{0}\cdot{1}".format(left, right) if right != "" else left

    def __generateSequence(self):
        stuff = []
        for (n, repeat) in self.repeatedNodes:
            for i in range(repeat):
                stuff.append(n.name)
        return stuff


def divDiffGenerator(nodes):
    """
```

```
    Generates DDs for specified nodes
    DD(x0)
    DD(x0,x0)
    DD(x0,x0,x0)
    DD(x0,x0,x0,x1)
    """
    dds = []
    current = []
    for n in nodes:
        current = current + [()]
        for i in range(n.multiplicity):
            current = current[:-1] + [(n, i+1)]
            dds.append(DivDifference(current))
    #print([dd.printTex() for dd in dds])
    #for dd in dds:
    #    print("{0} = {1}".format(dd.printFactor(), dd.calculate()))
    return dds

def Polynomial(poly):
    def inner(x):
        return sum([part.f(x) for part in poly])
    return inner

def make_tex(testcases, poly, self_name, filename="output.tex"):
    with open(filename, 'w') as outfile:
        outfile.write(tex_start)
        for i, p in enumerate(poly):
            outfile.write(r"\null\hrulefill\\"+"\n")
            outfile.write("Полином $H_{{{0}}}$:\\\\\n".format(i+1))
            outfile.write("\Узлын:\\\\\n")
            for tk in testcases[i]:
                outfile.write("{0} \\\\\\\\\n".format(tk.printTex()))
            parts = [x.printTex() for x in p]
            outfile.write("\n\\begin{gather*}\n")
            outfile.write("H_{{{0}}} = {1}\\\\\n".format(i+1, " + \\\\\n".joi

            outfile.write(r"\\")
            #now calculated values
            for part in p:
                symbolic = part.printFactor()
                value = part.calculate()
                outfile.write("{0} = {1}\\\\\n".format(symbolic, value))

            outfile.write("\\end{gather*}\\\\\n")
        outfile.write(tex_end.format(self_name))
```

```
def plot(poly, xmin, xmax):
    """
    Draw nice graphics
    """
    def get_name(i):
        if i==0:
            return "$f(x)$"
        else:
            return "$H_{{{0}}}$".format(i)
    functions = [f] + [Polynomial(p) for p in poly]

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)

    x = numpy.linspace(xmin, xmax)
    ax.grid(True, which='both')
    ax.xaxis.set_ticks(range(xmin, xmax))
    #ax.yaxis.set_ticks(range(-50, 50))
    plts = [tuple(plt.plot(x, function(x), linewidth=1))
     for function  in functions]

    names = [get_name(i) for i in range(len(plts))]
    ax.legend(plts, names, loc=4)

    #plt.show()
    plt.savefig('figure.png', bbox_inches=0)

def main():
    """
    Runner for Hermit calculator
    """
    xs = [
        ('x_0', 0),
        ('x_1', 5),
        ('x_2', 10),
    ]
    testCases = [
        [Node(xs[0], 3), Node(xs[1], 1), Node(xs[2], 1)],
        [Node(xs[0], 1), Node(xs[1], 2), Node(xs[2], 1)],
        [Node(xs[0], 1), Node(xs[1], 1), Node(xs[2], 2)],
#         [Node('x_{-5}', -5, 1), Node('x_3', 3, 1), Node('x_7', 7, 2), Node('
    ]

    poly = [divDiffGenerator(tk) for tk in testCases]
```

```
    xmin = min([x[1] for x in xs])
    xmax = max([x[1] for x in xs])

    plot(poly, xmin, xmax)
    make_tex(testCases, poly, argv[0])


tex_start = r"""\documentclass[12pt]{article}
\usepackage{amsmath}
\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}
\usepackage{graphicx}
\usepackage{hyperref}
\usepackage[margin=1in]{geometry}
\usepackage{listings}
\parindent=0cm

\usepackage{mathtools}
\DeclarePairedDelimiter{\abs}{\lvert}{\rvert}

\newtheorem{taskМногочлен}{ Эрмита}

\newcommand{\code}[2]{
  \hrulefill
  \subsection*{#1}
  \lstinputlisting{#2}
  \vspace{2em}
}

\begin{document}
    \titleЛабораторная{ работа}
    \authorКириллов{ PA.. @ КН-402}
    \maketitle
\begin{gather*}
    \textФункция{: } f(x) = \frac{5}{1+5e^{-0.5x}} \\\\
    f'(x) = \frac{12.5e^{0.5x}}{(5+e^{0.5x})^2} \\\\
    f''(x) = \frac{e^{-0.5x}\cdot(31.25e^x - 6.25e^{1.5x})}{(5 + e^{0.5x})^3}
\end{gather*}
"""
tex_end = r"""\lstset{{
  language=Python,
  showstringspaces=false,
  formfeed=\newpage,
  tabsize=4,
  commentstyle=\itshape,
```

```
%  basicstyle=\ttfamily,
  morekeywords={{models, lambda, forms}},
  inputencoding=utf8,
  extendedchars=\true
}}
  \includegraphics[]{{/home/rast/aperture/lab_final/figure.png}}
\codeКод{{ на python, использованныйдлявычислений : (\url{{github.com/Rast1234/

\end{{document}}"""

if __name__ == "__main__":
    main()
```