Fys4150: Introduction to

Project 1

Due date **19.rd of September, 2016 - 23:59**

Øyvind B. Svendsen, Magnus Christopher Bareid
un: oyvinbsv, magnucb

September 15, 2016

Abstract

The aim of this project is to get familiar with various vector and matrix operations, from dynamic memory allocation to the usage of programs in the library package of the course.

The student was invited to use either brute force-algorithms to calculate linear algebra, or to use a set of recommended linear algebra packages through Armadillo that simplify the syntax of linear algebra. Additionally, dynamic memory handling is expected.

The students will showcase necessary algebra to perform the tasks given to them, and explain the way said algebra is implemented into algorithms. In essence, we're asked to simplify a linear second-order differential equation from the form of the Poisson equation, seen as

$$\nabla^2 \Phi = -4\pi\rho(\mathbf{r})$$

into a one-dimensional form bounded by Dirichlet boundary conditions.

$$-u''(x) = f(x)$$

so that discretized linear algebra may be committed unto the equation.

Contents

1 Computational Physics: First project **3**
1.1 (a): The fundamental math . . . 3
1.1.1 Intro . . . 3
1.1.2 Method . . . 3
1.1.3 Results . . . 4
1.1.4 Discussion . . . 4
1.2 (b) . . . 4
1.2.1 Intro . . . 4
1.2.2 Method . . . 4
1.2.3 Results . . . 4
1.2.4 Discussion . . . 4
1.3 (c) . . . 4
1.3.1 Intro . . . 4
1.3.2 Method . . . 4
1.3.3 Results . . . 4
1.3.4 Discussion . . . 4
1.4 (d) . . . 4
1.4.1 Intro . . . 4
1.4.2 Method . . . 4
1.4.3 Results . . . 4
1.4.4 Discussion . . . 4
1.5 (e) . . . 4
1.5.1 Intro . . . 4
1.5.2 Method . . . 4
1.5.3 Results . . . 4
1.5.4 Discussion . . . 4

2 Appendix - Program list **5**

1 Computational Physics: First project

1.1 (a): The fundamental math

1.1.1 Intro

The production of this document will inevitably familiarize its authors with the programming language `C++`, and to this end mathematical groundwork must first be elaborated to translate a Poisson equation from continuous calculus form, into a discretized numerical form.

The Poisson equation is rewritten to a simplified form, for which a real solution is given, with with we will compare our numerical approximation to the real solution.

1.1.2 Method

Reviewing the Poisson equation:

$$\begin{aligned}
\nabla^2\Phi &= -4\pi\rho(\mathbf{r}), \text{ which is simplified one-dimensionally by } \Phi(r) = \phi(r)/r \\
\Rightarrow \frac{d^2\phi}{dr^2} &= -4\pi r\rho(r), \text{ which is further simplified by these substitutions:} \\
r &\rightarrow x, \\
\phi &\rightarrow u, \\
4\pi r\rho(r) &\rightarrow f, \qquad \text{which produces the simplified form}
\end{aligned}$$

$$\begin{aligned}
&-u''(x) = f(x), \qquad \text{for which we assume that} \qquad f(x) = 100e^{-10x}, \qquad (1) \\
&\Rightarrow u(x) = 1 - (1 - e^{-10})x - e^{-10x}, \text{ with bounds: } x \in [0, 1],\ u(0) = u(1) = 0
\end{aligned}$$

To more easily comprehend the syntax from a programming viewpoint, one may refer to the each discretized representation of x and u; we know the span of x, and therefore we may divide it up into appropriate chunks. Each of these x_i will yield a corresponding u_i.

We may calculate each to each discrete x_i by the form $x_i = ih$ in the interval from $x_0 = 0$ to $x_n = 1$ as it is linearly increasing, meaning we use n points in our approximation, yielding the step length $h = 1/n$. Of course, this also yields for the discretized representation of $u(x_i) = u_i$.

Through Euler's teachings on discretized numerical derivation methods, a second derivative may be constructed through the form of

$$-u''(x) = -\frac{u_{+1} + u_{i-1} - 2u_i}{h^2} = \frac{2u_i - u_{+1} - u_{i-1}}{h^2} = f_i, \qquad \text{for } i = 1, ..., n \qquad (2)$$

1.1.3 Results

1.1.4 Discussion

1.2 (b)

1.2.1 Intro

1.2.2 Method

1.2.3 Results

1.2.4 Discussion

1.3 (c)

1.3.1 Intro

1.3.2 Method

1.3.3 Results

1.3.4 Discussion

1.4 (d)

1.4.1 Intro

1.4.2 Method

1.4.3 Results

1.4.4 Discussion

1.5 (e)

1.5.1 Intro

1.5.2 Method

1.5.3 Results

1.5.4 Discussion

---

2 Appendix - Program list

This is the code used in this assignment. Anything that was done by hand has been implemented into this pdf, above. `plot_stuff.py`

```python
import pylab as pyl
import os
import sys

curdir  = os.getcwd()
version  = 3
data_dict = {}
column_dict = {}
n_range = [10,100,1000]

for n in n_range:
    #loop through different n's
    with open(curdir+"/data/dderiv_u_python_v%s_n%d.dat"%(version,n), 'r') as infile:
         full_file  = infile.read() #read entire file into text
        lines  = full_file.split('\n') #separate by EOL-characters
        lines  = lines[:-1] #remove last line (empty line)
        keys = lines.pop(0).split(', ') #use top line as keys for dict.
        dict_of_content = {}
        for i,key in zip(range(len(keys)), keys):
            #loop over keys: h, f2c, f3c
            dict_of_content[key] = []
            for j in range(len(lines)):
                # loop over all lines
                line  = lines[j].split(', ')
                word = line[i]
                try:
                    word = float(word)
                except ValueError: #word cannot be turned to number
                    print word
                    sys.exit("There is something wrong with your data-file \n'%s' cannot be turned to numbers"%word)
                dict_of_content[key].append(word)
        data_dict["n=%d"%n] = dict_of_content

def u_exact(x):
    u =  1.0 - (1.0 - pyl.exp(-10.0))*x - pyl.exp(-10.0*x)
    return u

def plot_generator(version, n):
    """
    plot generator of generated data
    """
     datafile  = open(curdir+"/data/dderiv_u_python_v%s_n%d.dat"%(version,n))
    data = []

    for line in datafile:
         linesplit  = [item.replace(",","") for item in line.split()]
        data.append(linesplit)

    columns = data[0]
    data     = pyl.array(data[1:]).astype(pyl.float64)
    for i in xrange(len(columns)-1):
        pyl.figure() # comment out this line to unify the plots ... when their dimensions correlate
        pyl.plot(data[:,0],  data[:,i+1], label=r"%s" % columns[i+1])
        pyl.xlabel("x")
        pyl.ylabel(r"%s" % columns[i+1])
```

```
        pyl.title(r"Plot\ of\ %s\ over\ x" % columns[i+1])
        pyl.legend(loc='best')

    # pyl.savefig("evil_plot.png", dpi=400)
    pyl.show()
    datafile.close()

def harry_plotter():
    #plot pre-game
    pyl.figure()
    pyl.grid(True)
    pyl.title("function u for different steplengths")
    pyl.ylabel("u(x)")
    pyl.xlabel("x")
    for n in n_range:
        x = pyl.array(data_dict["n=%d"%n]["x"])
        u_gen = pyl.array(data_dict["n=%d"%n]["u_gen"])
        u_spec = pyl.array(data_dict["n=%d"%n]["u_spec"])
        u_LU = pyl.array(data_dict["n=%d"%n]["u_LU"])
        pyl.plot(x,u_gen, 'g-', label="general tridiag, n=%d"%n)
        pyl.plot(x,u_spec, 'r-', label="specific tridiag, n=%d"%n)
        pyl.plot(x,u_LU, 'b-', label="LU-dekomp, n=%d"%n)

    u_ex = u_exact(x)
    pyl.plot(x, u_ex, 'k-', label="exact, n=%d"%len(x))
    #pyl.legend(loc="best",prop={"size":8})
    pyl.show()
    return None

def compare_methods(n):
    """
    For a specific length 'n' compare all three methods
    with the exact function.
    """
    x = pyl.array(data_dict["n=%d"%n]["x"])
    gen = pyl.array(data_dict["n=%d"%n]["u_gen"])
    spec = pyl.array(data_dict["n=%d"%n]["u_spec"])
    LU = pyl.array(data_dict["n=%d"%n]["u_LU"])
    exact = u_exact(x)
    pyl.figure("compare methods")
    pyl.grid(True)
    pyl.hold(True)
    pyl.xlabel("x")
    pyl.ylabel("u(x)")
    pyl.title("function u for three different methods (n=%d)"%n)

    pyl.plot(x, exact, 'k-', label="exact")
    pyl.plot(x, gen, 'b-', label="general tridiagonal")
    pyl.plot(x, spec, 'g-', label="specific tridiagonal")
    pyl.plot(x, LU, 'r-', label="LU-decomp.")
    pyl.legend(loc='best', prop={'size':9})
    pyl.savefig(curdir+"/img/compare_methods_n%d.png"%n)

def compare_approx_n(n_range=[10,100,1000], approx_string="general"):
    """
    For all n's available, plot the general approximation and
    exact solution
    """
    if approx_string == "general":
        approx_key = "u_gen"
```

```
    elif approx_string == "specific":
        approx_key = "u_spec"
    else:
        sys.exit("In function 'compare_approx_n', wrong argument 'approx_string'")
    pyl.figure("compare %s"%approx_string)
    pyl.grid(True)
    pyl.hold(True)
    pyl.xlabel("x")
    pyl.ylabel("u(x)")
    pyl.title("approximation by %s tridiagonal method"%approx_string)

    for n in n_range:
        x = pyl.array(data_dict["n=%d"%n]["x"])
        u_approx = pyl.array(data_dict["n=%d"%n][approx_key])
        pyl.plot(x, u_approx, '--', label="n=%1.1e"%n)

    x = pyl.linspace(0,1,1001)
    exact = u_exact(x)
    pyl.plot(x, exact, '-', label="exact")
    pyl.legend(loc='best', prop={'size':9})
    pyl.savefig(curdir+"/img/compare_%s_n_n%d.png"%(approx_string,n))

def epsilon_plots(n_range=[10,100,1000]):
    eps_max = pyl.zeros(len(n_range))
    h = pyl.zeros(len(n_range))
    for i, n in enumerate(n_range):
        x = pyl.array(data_dict["n=%d"%n]["x"])
        u = u_exact(x)
        v = pyl.array(data_dict["n=%d"%n]["u_gen"])
        #remove edges of u and v to avoid 'divide-by-zero'
        u = u[1:-1]; v = v[1:-1]
        eps = pyl.log10(abs((v-u)/u))
        eps_max[i] = max(eps)
        #sys.exit()
        h[i] = pyl.log10(1.0/(n+1))
    pyl.figure("epsilon")
    pyl.grid(True)
    pyl.hold(True)
    pyl.xlabel(r"\log_{10}(h)")
    pyl.ylabel(r"$\epsilon = \log_{10}\left(\frac{u_{approx}-e_{exact}}{u_{exact}}\right)$")
    pyl.title("log-plot of epsilon against step-length h")
    pyl.plot(h, eps_max, 'ko')
    pyl.legend(loc='best')
    pyl.savefig(curdir+"/img/epsilon.png")

#make plots
#compare_methods(n=1000)
#compare_approx_n(approx_string="general")
compare_approx_n(approx_string="specific")
pyl.show()
#epsilon_plots()
#pyl.show()
```