Numerical integration using linear algebra

Anders P. Åsbø
(Dated: September 9, 2019)

derp

CONTENTS

I. INTRODUCTION

II. FORMALISM

III. IMPLEMENTATION

IV. ANALYSIS

V. CONCLUSION

Appendix A: Program files

All code for this report was written in Python 3.6, and the complete set of files can be found at https://github.com/FunkMarvel/CompPhys-Project-1.

1. project.py

```
# Project 1 FYS3150, Anders P. Åsbø
# general tridiagonal matrix.
import data_generator as gen
import numpy as np
import os
import matplotlib.pyplot as plt
import timeit as time


def main():
    """Program solves matrix equation Au=f, using decomposition, forward
    substitution and backward substitution, for a tridiagonal, NxN matrix A."""
    init_data()  # initialising data

    # performing decomp. and forward and backward sub.:
```

```python
    decomp_and_forward_and_backward_sub()

    save_sol()  # saving numerical solution in "data_files" directory.

    plot_solutions()  # plotting numerical solution vs analytical solution.

    plt.show()  # displaying plot.


def init_data():
    """Initialising data for program as global variables."""
    global dir, N, name, x, h, anal_sol, u, d, d_prime, a, b, g, g_prime
    dir = os.path.dirname(os.path.realpath(__file__))  # current directory.

    # defining number of rows and columns in matrix:
    N = int(eval(input("Specify number of data points N: ")))
    # defining common label for data files:
    name = input("Label of data-sets without file extension: ")

    x = np.linspace(0, 1, N)  # array of normalized positions.
    h = (x[0]-x[-1])/N  # defining step-siz.

    gen.generate_data(x, name)  # generating dataanal_name set.
    anal_sol = np.loadtxt("%s/data_files/anal_solution_for_%s.dat" %
                          (dir, name))

    u = np.empty(N)  # array for unkown values.
    d = np.full(N, 2)  # array for diagonal elements.
    d_prime = np.empty(N)  # array for diagonal after decom. and sub.
    a = np.full(N-1, -1)  # array for upper, off-center diagonal.
    b = np.full(N-1, -1)  # array for lower, off-center diagonal.
    # array for g in matrix eq. Au=g.
    f = np.loadtxt("%s/data_files/%s.dat" % (dir, name))
    g = f*h**2
    g_prime = np.empty(N)  # array for g after decomp. and sub.


def decomp_and_forward_and_backward_sub():
    """Function that performs the matrix decomposition and forward
    and backward substitution."""
    # setting boundary conditions:
    u[0], u[-1] = 0, 0
    d_prime[0] = d[0]
    g_prime[0] = g[0]

    start = time.default_timer()
    for i in range(1, len(u)):  # performing decomp. and forward sub.
        decomp_factor = b[i-1]/d_prime[i-1]
        d_prime[i] = d[i] - a[i-1]*decomp_factor
        g_prime[i] = g[i] - g_prime[i-1]*decomp_factor

    for i in reversed(range(1, len(u)-1)):  # performing backward sub.
        u[i] = (g_prime[i]-a[i]*u[i+1])/d_prime[i]
    end = time.default_timer()
    print("Time spent on loop %e" % (end-start))


def save_sol():
    """Function for saving numerical solution in data_files directory
    with prefix "solution"."""
    path = "%s/data_files/solution_%s.dat" % (dir, name)
    np.savetxt(path, u, fmt="%f")


def plot_solutions():
    """Function for plotting numerical vs analytical solutions."""
    x_prime = np.linspace(x[0], x[-1], len(anal_sol))

    plt.figure()
    plt.plot(x, u, label="Numerical solve")
    plt.plot(x_prime, anal_sol, label="Analytical solve")
    plt.title("Integrating with a %iX%i tridiagonal matrix" % (N, N))
    plt.xlabel(r"$x \in [0,1]$")
    plt.ylabel(r"$u(x)$")
    plt.legend()
    plt.grid()


if __name__ == '__main__':
    main()


# example run:
```

```
"""
$ python3 project.py
Specify number of data points N: 1000
Label of data-sets without file extension: num1000x1000
"""
# a plot is displayed, and the data is saved to the data_files directory.
```

2. project__specialized.py

```
# Project 1 FYS3150, Anders P. Åsbø
import data_generator as gen
import numpy as np
import os
import matplotlib.pyplot as plt
import timeit as time


def main():
    """Program solves matrix equation Au=f, using decomposition, forward
    substitution and backward substitution, for a Toeplitz, NxN matrix A."""
    init_data()  # initialising data

    # performing decomp. and forward and backward sub.:
    decomp_and_forward_and_backward_sub()

    save_sol()  # saving numerical solution in "data_files" directory.

    plot_solutions()  # plotting numerical solution vs analytical solution.

    plt.show()  # displaying plot.


def init_data():
    """Initialising data for program as global variables."""
    global dir, N, name, x, h, anal_sol, u, d, d_prime, a, b, g, g_prime
    dir = os.path.dirname(os.path.realpath(__file__))  # current directory.

    # defining number of rows and columns in matrix:
    N = int(eval(input("Specify number of data points N: ")))
    # defining common label for data files:
    name = input("Label of data-sets without file extension: ")

    x = np.linspace(0, 1, N)  # array of normalized positions.
    h = (x[0]-x[-1])/N  # defining step-siz.

    gen.generate_data(x, name)  # generating dataanal_name set.
    anal_sol = np.loadtxt("%s/data_files/anal_solution_for_%s.dat" %
                          (dir, name))

    u = np.empty(N)  # array for unkown values.
    s = np.arange(1, N+1)
    d_prime = 2*(s)/(2*(s+1))  # pre-calculating the 1/d_prime factors.
    f = np.loadtxt("%s/data_files/%s.dat" % (dir, name))
    g = f*h**2
    g_prime = np.empty(N)  # array for g after decomp. and sub.


def decomp_and_forward_and_backward_sub():
    """Function that performs the matrix decomposition and forward
    and backward substitution."""
    # setting boundary conditions:
    u[0], u[-1] = 0, 0
    g_prime[0] = g[0]
    start = time.default_timer()
    for i in range(1, len(u)):  # performing decomp. and forward sub.
        g_prime[i] = g[i] + g_prime[i-1]*d_prime[i-1]

    for i in reversed(range(1, len(u)-1)):  # performing backward sub.
        u[i] = (g_prime[i] + u[i+1])*d_prime[i-1]

    end = time.default_timer()
    print("Time spent on loop %e" % (end-start))


def save_sol():
    """Function for saving numerical solution in data_files directory
    with prefix "solution"."""
    path = "%s/data_files/solution_%s.dat" % (dir, name)
    np.savetxt(path, u, fmt="%f")
```

```python
def plot_solutions():
    """Function for plotting numerical vs analytical solutions."""
    x_prime = np.linspace(x[0], x[-1], len(anal_sol))

    plt.figure()
    plt.plot(x, u, label="Numerical solve")
    plt.plot(x_prime, anal_sol, label="Analytical solve")
    plt.title("Integrating with a %iX%i tridiagonal matrix" % (N, N))
    plt.xlabel(r"$x \in [0,1]$")
    plt.ylabel(r"$u(x)$")
    plt.legend()
    plt.grid()


if __name__ == '__main__':
    main()


# example run:
"""
$ python3 project_specialized.py
Specify number of data points N: 1000
Label of data-sets without file extension: opti1000x1000
"""
# a plot is displayed, and the data is saved to the data_files directory.
```

3. data_generator.py

```python
# create data set for numerical testing
import numpy as np
import os

dir = os.path.dirname(os.path.realpath(__file__))


def main():
    test_generate_data()
    test_generate_tridiagonal()


def generate_data(x, name):
    """Function that generates a set of u''(x) data, as well as the
    corresponding, analytical u(x). The data is saved to text"""
    data = 100*np.exp(-10*x)
    path = "%s/data_files/%s.dat" % (dir, name)
    np.savetxt(path, data, fmt="%f")

    x_prime = np.linspace(x[0], x[-1], 1000)
    analytical_solution = 1-(1-np.exp(-10))*x_prime-np.exp(-10*x_prime)
    analytical_solution[0], analytical_solution[-1] = 0, 0
    anal_name = "%s/data_files/anal_solution_for_%s.dat" % (dir, name)
    np.savetxt(anal_name, analytical_solution, fmt="%f")


def generate_tridiagonal(N):
    """Function that generates a Nx3 array with each column corresponding to
    the non-zero elements in a tridiagonal matrix.
    "b" (mat_data[:,0]) is the lower diagonal,
    "d" (mat_data[:,1]) is the diagonal,
    and "a" (mat_data[:,2]) is the upper diagonal."""
    mat_data = np.random.randint(1, 100, size=(N, 3))
    np.savetxt("b-d-a_tridiagonal.dat", mat_data, fmt="%f")


def test_generate_data():
    """Generates test data if run as stand-alone program."""
    x = np.linspace(0, 1, 1001)
    test_name = "Test_data"
    generate_data(x, test_name)


def test_generate_tridiagonal():
    """Generates test data for tridiagonal."""
    generate_tridiagonal(100)
    print(np.loadtxt("b-d-a_tridiagonal.dat"))
```

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