University of Zurich - HS 2013

Computational Science I
Exercise notes: Legendre, Chebyshev and Gauss Quadrature

Tobias Grubenmann

November 2, 2013

Exercise 1

The following code generates the first 6 Legendre polynomials using the five point integration rule:

```
from FivePoint import fivepoint
from Evaluate import evalp

N = 6
points = 100
B = 50

polynomial = [[0 for x in range(N)] for x in range(N)]

polynomial[0][0] = 1

for i in range(1, N):

    polynomial[i][i] = 1

    # Orthogonalization
    for j in range(i):

        f = lambda x: evalp(polynomial[i], x) * evalp(
           polynomial[j], x)
        integral1 = fivepoint(f, -1, 1, B)

        f = lambda x: evalp(polynomial[j], x) * evalp(
           polynomial[j], x)

        integral2 = fivepoint(f, -1, 1, B)

        for k in range(j+1):

            polynomial[i][k] = polynomial[i][k] - integral1
               /integral2*polynomial[j][k]

for i in range(2, N):
    # Normalization
    f = lambda x: evalp(polynomial[i], x) * evalp(
       polynomial[i], x)
    integral = fivepoint(f, -1, 1, B)

    for j in range(i+1):

        polynomial[i][j] = polynomial[i][j]/sqrt(integral)
           *1.0/sqrt(i+0.5)

values = [[0 for x in range(100)] for x in range(N)]

for i in range(N):
    for j in range(points):
        values[i][j] = [(j-points/2.0)/(points/2.0), evalp(
           polynomial[i],  (j-points/2.0)/(points/2.0))]

for i in range(N):
    plot(values[i])
```

The code produces the following output:

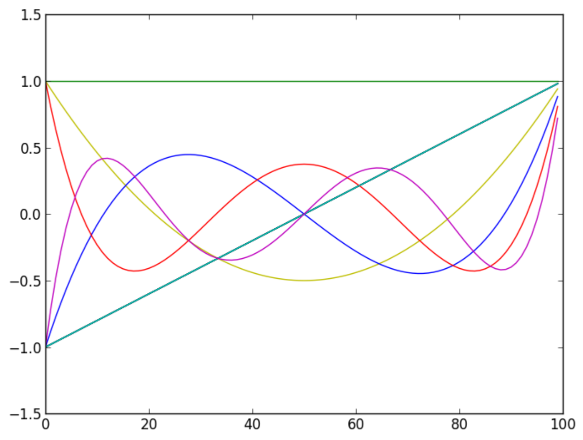

Figure 1: The first 6 Legendre polynomials

I used 50 blocks for the integration. The first 3 polynomials would even be accurate with only 1 block since we need to integrate the square of the polynomials and the integration rule is exact up to degree 5.

Exercise 2

The following code generates the first 6 Chebyshev polynomials using the five point integration rule:

```
from FivePoint import fivepoint
from Evaluate import evalp

N = 6
points = 100
B = 50

polynomial = [[0 for x in range(N)] for x in range(N)]

polynomial[0][0] = 1

for i in range(1, N):

    polynomial[i][i] = 1

    # Orthogonalization
    for j in range(i):
        f = lambda x: evalp(polynomial[i], x) * evalp(
           polynomial[j], x) / sqrt(1 - x**2)

        integral1 = fivepoint(f, -1, 1, B)

        f = lambda x: evalp(polynomial[j], x) * evalp(
           polynomial[j] , x) / sqrt(1 - x**2)

        integral2 = fivepoint(f, -1, 1, B)

        for k in range(j+1):

            polynomial[i][k] = polynomial[i][k] - integral1
               /integral2*polynomial[j][k]

for i in range(2, N):
    # Normalization
    f = lambda x: evalp(polynomial[i], cos(x)) * evalp(
       polynomial[i], cos(x))
    integral = fivepoint(f, 0, pi, B)

    for j in range(i+1):

        polynomial[i][j] = polynomial[i][j]/sqrt(integral)*
           sqrt(pi/2.0)

values = [[0 for x in range(100)] for x in range(N)]

for i in range(N):
    for j in range(points):
        values[i][j] = [(j-points/2.0)/(points/2.0), evalp(
           polynomial[i],  (j-points/2.0)/(points/2.0))]

for i in range(N):
    plot(values[i])
```

The code produces the following output:

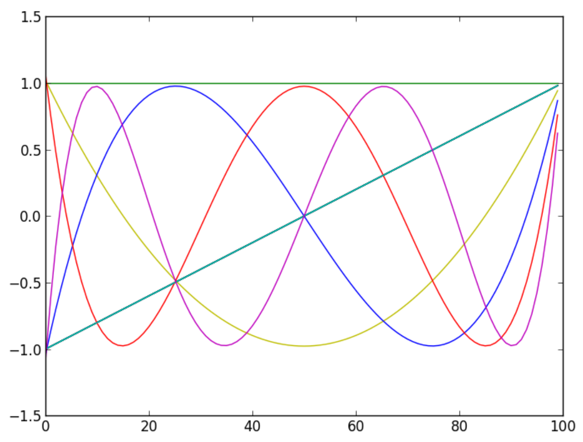

Figure 2: The first 6 Chebyshev polynomials

Exercise 3

The following codes calculates the Gauss-Chebyshev quadrature for a given function f using M points.

```
from math import *

def quadChebyshev( f , a, b, M = 5) :
# Compute integral of f between a and b
# using B blocks of the M-point integrator.

    integral = 0


    for j in range(M):

        integral += (b-a)/2.*pi/M*f((b-a)/2. * cos((j+0.5)*
           pi/M) + (a+b)/2.)

    return integral
```

The following code generates the first 6 Chebyshev polynomials using the Gauss-Chebyshev integration rule:

```
from GaussChebyshevQuadrature import quadChebyshev
from Evaluate import evalp
```

```
N = 6
points = 100
M = 10

polynomial = [[0 for x in range(N)] for x in range(N)]

polynomial[0][0] = 1

for i in range(1, N):

    polynomial[i][i] = 1

    # Orthogonalization
    for j in range(i):

        f = lambda x: evalp(polynomial[i], x) * evalp(
            polynomial[j], x)

        integral1 = quadChebyshev(f, -1, 1, M)

        f = lambda x: evalp(polynomial[j], x) * evalp(
            polynomial[j] , x)

        integral2 = quadChebyshev(f, -1, 1, M)

        for k in range(j+1):

            polynomial[i][k] = polynomial[i][k] - integral1
                /integral2*polynomial[j][k]

for i in range(2, N):
    # Normalization
    f = lambda x: evalp(polynomial[i], x) * evalp(
        polynomial[i], x)
    integral = quadChebyshev(f, -1, 1, M)

    for j in range(i+1):

        polynomial[i][j] = polynomial[i][j]/sqrt(integral)*
            sqrt(pi/2.0)

values = [[0 for x in range(100)] for x in range(N)]
```

```
for i in range(N):
    for j in range(points):
        values[i][j] = [(j-points/2.0)/(points/2.0), evalp(
            polynomial[i],  (j-points/2.0)/(points/2.0))]

for i in range(N):
    plot(values[i])
```

The code produces the following output:

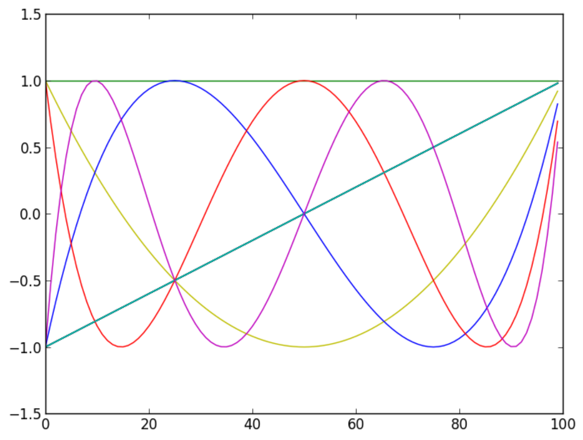

Figure 3: The first 6 Chebyshev polynomials

We can see at the intersection points of the polynomials, that the Gauss-Chebyshev quadrature delivers more accurate results even with only 10 evaluation points. This is due to the fact that the quadrature has order 21 (2N+1) which means that the results are correct up to order 20.