Numerical Approximation
NUMA12, Spring, 2017
Homework 5

Anton Makarov and Emil Johansson
Lund University

Task 1. Write a program for the exchange algorithm. The program should take as input:

- a continuous function f which is to be approximated
- the dimension n of the subspace A
- the interval $[a, b]$ in which f should be considered
- a reference (ordered set of $n + 1$ points in $[a, b]$)
- optionally a basis $\Phi_i\,, i = 1, \ldots, n$ of the Haar space (i.e. a set of n functions). If this input is not given your program should assume A = P n1 and use the monomial basis instead.
- a tolerance tol.
- a number nsp of sample points (see below)

The program needs to evaluate the max-norm of the error. For this end, the error function has to be sampled at nsp equidistant points. The program should return

- the coefficients of the approximation to the best approximation
- an estimate of the error (distance to the best approximation in the max-norm)
- the number of iterates
- a vector with the reference level values h for every iteration
- the actual reference

Test your program with the Runge function $f = \frac{1}{1+25x^2}\,, [a, b] = [-1, 1]$ and $A = P_{n-1}$ for various v.

Solution. This is our attempt at a exchange algorithm. The idea was to be "smart" and use a symbolic representation for the functions. But it generates mostly sorrow and bugs. Time however ran out so this is what we have.

```
## Task 1: The exchange algorithm


from sympy import symbols, lambdify, sin
import numpy as np
```

```
from matplotlib import pyplot as plt

def sign(x):
    if x <= 0:
        return -1
    if x > 0:
        return 1

def hcoeff(x):
    try:
        res = [];
        for k in x:
            res.append((-1)**k);
        return res;
    except TypeError:
        return (-1)**k;



def polynom(x, basis, coef):
    try:
        res = np.zeros(len(x));
        for i in range(len(coef)):
            for j in range(len(x)):
                res[j] += coef[i] * basis(x[j], i);
        return res;
    except TypeError: # x not list but number.
        res = 0;
        for i in range(len(coef)):
                res[j] += coef[i] * basis(x, i);
        return res;

def error(f, x, basis, coef):
    return f(x) - polynom(x, basis, coef);



def exchange_algorithm (f, n, intervall, reference, tolerance, nsp, basis=pow):
```

```
if (len(reference) != n + 1):
    raise ValueError("Reference must have exactly n+1 elements")

a, b = intervall

matrix = np.zeros([n+1, n+1])
grid = np.linspace(a, b, num=nsp)
h = [];
count = 0;

legend = []; # for plotting

while ( (len(h) < 2
         or (abs(h[len(h)-1] - h[len(h) - 2]) > tolerance))
        and count < 10):
    count += 1
    # Set up matrix from reference
    for i in range(n):
        for j in range(n+1):
            matrix[i][j]= basis(reference[j], i)
    matrix[n] = hcoeff(np.array(range(n+1)))

    #calculate coefficients and h
    #res =  np.linalg.inv(np.transpose(matrix)).dot(f(reference))
    res = np.linalg.solve(np.transpose(matrix), f(reference))
    h.append(res[n])

    # Find the max
    coefficients = np.delete(res, -1);
    error_grid = error(f, grid, basis, coefficients);
    error_ref = error(f, reference, basis, coefficients);
    max_index = np.argmax(np.abs(error_grid));
    maxpos = grid[max_index];
    max_error = error_grid[max_index];
    plt.plot(grid, error_grid);
    #plt.plot(reference, np.zeros(len(reference)), "*")
```

```
        legend.append(str(count))

        #Case: Before first
        if (maxpos < reference[0]):
            if (sign(max_error) == sign(error_ref[0])):
                reference[0] = maxpos
            else:
                reference = np.delete(reference, n)#delete last element
                reference = np.insert(reference, 0, maxpos) #insert maxpos as first

        #Case: between two reference points
        for i in range(len(reference)-1):
            if ( (maxpos > reference[i]) and (maxpos < reference[i+1]) ):
                if (sign(max_error) == sign(error_ref[i])):
                    reference[i] = maxpos
                else:
                    reference[i+1] = maxpos

        #Case: After last point
        if (maxpos > reference[n]):
            if (sign(max_error) == sign(error_ref[n])):
                reference[n] = maxpos
            else:
                reference = np.delete(reference, 0) #delete first element.
                reference = np.append(reference, maxpos) # append with new last element.
    plt.legend(legend)
    plt.savefig("error.png")
    plt.figure(2);
    plt.plot(grid, f(grid), grid, polynom(grid, basis, coefficients))
    plt.legend(["f", "approx"])
    plt.savefig("approx.png")
    return coefficients, np.abs(max_error), count, h, reference;

if __name__ == "__main__":
    def f(x):
        try:
```

```
            return 1 / (1 + 25*x**2);
        except TypeError:
            res = [];
            for y in x:
                res.append(1 / (1 + 25*x**2));
            return res;

x = symbols('x')
n = 40
intervall = (0, 1)
reference = np.linspace(0, 1, n+1)
tolerance = 10**(-10)
nsp = 100
coef, error, count, h, reference = exchange_algorithm(f, n, intervall, reference, tolerance, nsp)
print("Coefficients: " + str(coef))
print("error: " + str(error))
print("count: " + str(count))
print("h: " + str(h))
print("reference: " + str(reference))
```

■

Task 2. Find the best minimax approximation from the space of polynomials of degree at most one to $f(x) = x^2\,, x \in [0, 1]$. Plot the result together with the original function. Plot the error function.

Solution. Theorem 7.4 gives us that if we find a function with 3 points in the interval where the error equals the maximum error then we have our best approximation. From figure 2 it is clear that the function $x - 1/8$ in figure 1 that was found in an earlier homework satisfies the criteria of theorem 7.4 and is this a best approximation.

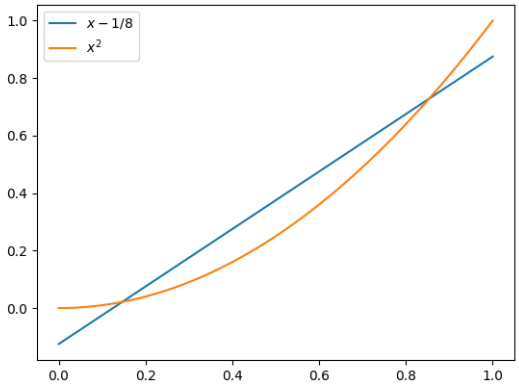

Figure 1: $x - 1/8$ and x^2

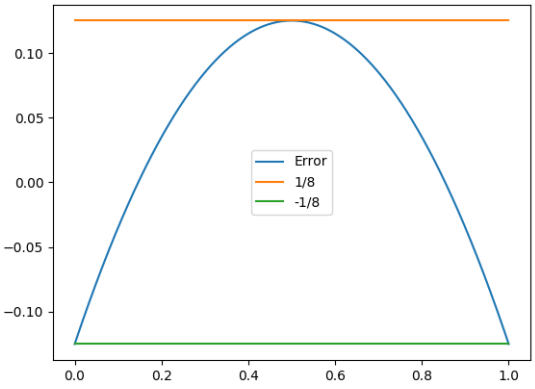

Figure 2: $x - 1/8 - x^2$

■

Task 3. Use a discrete version of the one-point exchange algorithm to calculate the best minimax approximation from the space of polynomials of degree at most one to the following seven function values: $f(0) = 0.3$, $f(1) = 4.2$, $f(2) = 0.1$, $f(3) = 3.4$, $f(4) = 5.7$, $f(5) = 4.9$, $f(6) = 5.7$. Let the initial reference be the set of points $\{0, 3, 6\}$.

Solution. When we are given a set of functional values on a grid:

$$\mathcal{G} = \{x_1, x_2, \ldots, x_m\} \subset [a, b] \quad m > dim(\mathcal{A})$$

we modify the method with $\mathcal{Z} \subset \mathcal{G}$. Let us consider the monomial basis $\{1, x\}$ and solve the equation system:

$$f(\xi_i) = \lambda_0 + \lambda_1 \xi_i + (-1)^i h$$

On the first iteration we obtain the solution $p(x) = 0.5 + 0.9x$ with $h = -0.2$ and change the reference to $\{0, 1, 6\}$ (the closest point ξ of the error function with the same sign as η). On the fourth iteration we finally arrive at the best minimax approximation (the discrete algorithm always finds the best minimax approximation in a finite number of steps), that is $p(x) = 1.4 + 0.5x$ with $h = 2.3$ on the reference $\{1, 2, 4\}$ as we can see in Figure 3. ■

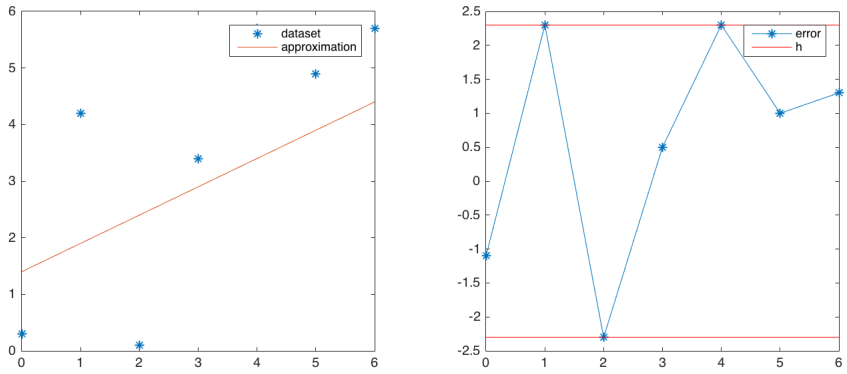

Figure 3: plot of the dataset approximated by $1.4 + 0.5x$, error and h. Reference $\{1, 2, 4\}$

Task 4. Find the best least squares approximation from the space of polynomials of degree at most one to $f(x) = x^2$, $x \in [0, 1]$. Plot the result together with the original function. Plot the error function. Calculate $\|f - p\|$ and $\|p^*\|$.

Solution. We know from the characterization theorem that given a linear subspace ($\mathcal{A} \equiv \mathcal{P}_1$) of a Hilbert space ($\mathcal{B} \equiv \mathcal{C}[0,1]$) then $p^* \in \mathcal{A}$ is the best least squares approximation to $f \in \mathcal{B}$ if and only if the error $e^* = f - p^*$ satisfies:

$$\langle e^*, p \rangle = 0, \quad \forall p \in \mathcal{A}$$

As we can write the polynomial as $p^* = \sum_{i=0}^{n} c_i^* \Phi_i$ where Φ_i are the basis functions we have that to be the best approximation we need to satisfy $\langle \Phi_i, f - p^* \rangle = 0$ that is:

$$\sum_{j=0}^{n} \langle \Phi_i, \Phi_j \rangle c_j^* = \langle \Phi_i, f \rangle$$

this expression is called normal equations. Then we have the theorem that says if $\mathcal{A}$ is a linear subspace of a Hilbert space $\mathcal{B}$ spanned by $\{\Phi_0 \dots \Phi_n\}$ where $\langle \Phi_i, \Phi_j \rangle = 0$ if $i \neq j$ then the least squares best approximation from $\mathcal{A}$ to $f \in \mathcal{B}$ is:

$$p^*(x) = \sum_{j=0}^{n} \frac{\langle \Phi_j, f \rangle}{\|\Phi_j\|_2^2} \Phi_j \tag{1}$$

And to make the basis orthogonal, we use the Gramm-Schmidt method. Let us now apply this two theorems to our function $f(x) = x^2$, $x \in [0,1]$. Take the basis functions $\{1, x\}$ and make them orthogonal:

$$\Phi_0 = 1$$
$$\Phi_1 = x\Phi_0 - \alpha_0 \Phi_0$$

Where $\alpha_0 = \frac{\langle \Phi_0, x\Phi_0 \rangle}{\|\Phi_0\|_2^2}$. This gives us that $\Phi_1 = (x - 1/2)$. Now let us compute the coefficients:

$$c_0^* = \frac{\langle 1, x^2 \rangle}{\|1\|_2^2} = \frac{1}{3}$$
$$c_1^* = \frac{\langle x - 1/2, x^2 \rangle}{\|x - 1/2\|_2^2} = 1$$

When we plug in this data into equation 1 we get that our best approximation polynomial is $p^*(x) =$

$x - 1/6$. we can see this in Figure 4. Finally we are asked to compute the norm of the error:

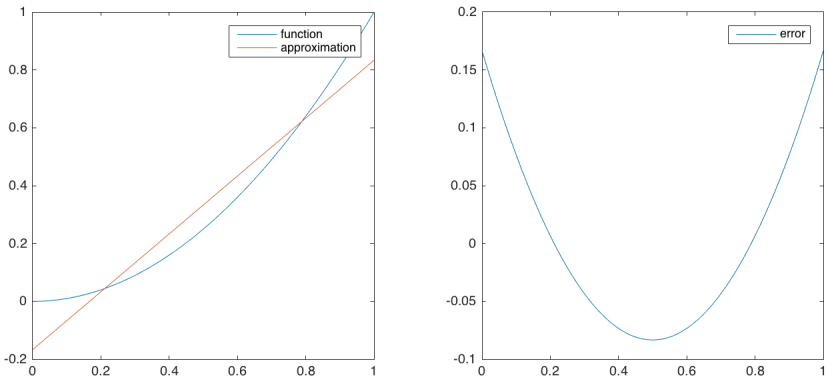

Figure 4: plot of the function x^2 approximated by $x - 1/6$ and error

$$\|x^2 - x + 1/6\|_2 = \sqrt{\int_0^1 (x^2 - x + 1/6)^2 \, dx} = \frac{\sqrt{5}}{30}$$

and the norm of p^*

$$\|x - 1/6\|_2 = \sqrt{\int_0^1 (x - 1/6)^2 \, dx} = \frac{\sqrt{7}}{6}$$

■