MTMM.00.005 Numerical Methods
Practical lesson 7: Ordinary iteration method for linear systems.

March 26, 2019

To get a positive result, student has to solve all of the exercises given in practical lessons, explain the programming code and answer the questions.

The deadline for the seventh set of exercises is April 2nd.

NB! In order to use the nonstandard packages as *numpy* and *matplotlib* one could use Command Prompt and type

```
H:\nm> "C:\Program Files\Anaconda3\python" <filename>.py
```

Here `H:\nm` is the folder for `<filename>.py`.

Ordinary iteration method for linear systems

Consider the system

$$\begin{cases} x_1 = b_{11}x_1 + \ldots + b_{1n}x_n + b_1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = b_{n1}x_1 + \ldots + b_{nn}x_n + b_n \end{cases}$$

or

$$x = Bx + b,$$

where $x = (x_j)$ is the vector of unknowns, $B = (b_{ij})$ is the matrix of system coefficients, and $b = (b_i)$ the vector of constant terms. The ordinary iteration method requires one initial value x^0, further we find

$$x^{m+1} = Bx^m + b, \quad m = 0, 1, \ldots .$$

Theorem 1 (sufficient condition for convergence). If

$$\max_{1\le i\le n} \sum_{j=1}^{n} |b_{ij}| < 1 \quad \text{or} \quad \max_{1\le j\le n} \sum_{i=1}^{n} |b_{ij}| < 1 \quad \text{or} \quad \sum_{i,j=1}^{n} b_{ij}^2 < 1$$

(i.e. $\|B\| < 1$), then the ordinary iteration method converges for any initial value to the unique solution of the system $x = Bx + b$.

Theorem 2 (necessary and sufficient condition for convergence). The ordinary iteration method for solving the system $x = Bx + b$ converges to the unique solution for any initial value if and only if the moduli of all eigenvalues of matrix B are smaller than 1, i.e., $\det(B - \lambda I) = 0 \Rightarrow |\lambda| < 1$.

Fredholm integral equation of the second kind and its solution by quadrature method

Consider the Fredholm integral equation of the second kind

$$u(x) = \int_a^b K(x,s)u(s)\,ds + f(x), \quad x \in [a,b].$$

Here the kernel K and the function f are given, a and b are constants. The problem is to find the function u.

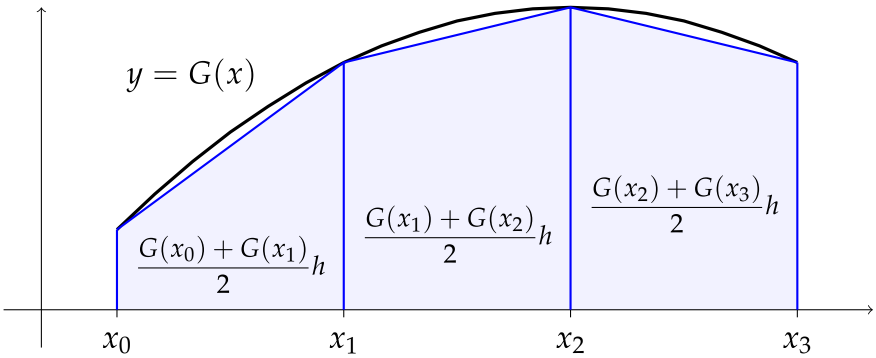
Take some n, uniform knots $a = x_0 < x_1 < \ldots < x_n = b$, where $x_i = a + ih$, $i = 0, \ldots, n$, and $h = \dfrac{b-a}{n}$.

Replace the integral with some quadrature formula, i.e.,

$$\int_a^b G(s)\, ds \approx \sum_{j=0}^{n} w_j\, G(x_j).$$

For example, take the trapezoidal formula, where

$$w_0 = \frac{h}{2}, \quad w_1 = h, \quad \ldots, \quad w_{n-1} = h, \quad w_n = \frac{h}{2}.$$



Now write the initial equation at knots $x_0, \ldots, x_n$ and replace the integral by its quadrature sum. We obtain the system of linear equations with respect to the unknowns $u_i = u(x_i)$, $i = 0, \ldots, n$,

$$u_i = \sum_{j=0}^{n} w_j K(x_i, x_j) u_j + f(x_i), \quad i = 0, \ldots, n. \tag{1}$$

The approximation of u at knots x_i is the solution of this linear system, i.e., $u(x_i) \approx u_i$. In general, the approximate values of u could be computed as

$$u(t) = \sum_{j=0}^{n} w_j K(t, x_j) u_j + f(t), \quad t \in [a, b].$$

How to measure the computing time

To measure the computing time use, for example, the package *timeit*. See the next example:

```
from timeit import default_timer as timer
start=timer()
for i in range (100):
    print(i)
end=timer()
print('Time elapsed:',end - start)
```

Exercise. Use quadrature method to solve the integral equation

$$u(x) = \int_0^{2\pi} \sin(x + 2s) u(s)\, ds + x, \quad x \in [0, 2\pi].$$

To solve the linear system (1) use:

a) the inverse matrix with `numpy.linalg.inv`,

b) the solver `numpy.linalg.solve` (which actually uses the Gaussian elimination),

c) ordinary iteration method.

Compare the computing time of these three methods in case of $n = 1000$, $n = 2000$, $n = 4000$. For the iteration method print out the number of iterations needed. Terminate the iteration process if $\|x^{m+1} - x^m\| \leq 10^{-7}$.

But first, write out the components of system (1), choose some small n and perform the iteration process. Draw the graph of approximate solution (i.e. the points (x_i, u_i), $i = 0, \ldots, n$, and the broken line joining them) and the graph of exact solution $u(x) = x - \pi \cos(x)$. Solve the same linear system by inverse matrix and `linalg.solve`, check the solutions by comparing the graphs of approximate and exact solutions. If all three methods give correct results, start to increase n and measure the time needed for computations.