MA2501 Numerical Methods
Spring 2018

Semester Project

Norwegian University of Science and Technology
Department of Mathematical Sciences

Practical Information

This project counts for 30 % of the final grade in the course.
The deadline for the project is **April 27 2018, at 23:59**.

The report and the codes should be sent electronically to Abdullah Abdulhaque via email (abdullah.abdulhaque@ntnu.no). Collect all the material in a compressed zip-file and mark it with your student numbers, not your names. You can work in groups of 3-4 members.

Some hints:

- Before starting, make sure that you master PYTHON well. We recommend the books and internet sites posted on the homepage.
- Read carefully through the *whole* appendix before solving the exercises. It contains a lot of necessary information required for the programming.
- Remember to use all the code templates listed on the project's webpage.
- Start as simple as possible. Do one thing at the time and verify that it is correct before proceeding. Compare with hand calculations on simple problems, if you find this easier. It is recommended to construct small reference problems and test them to ensure that the code is correct.
- You should hand in a written answer to all the questions in LaTeX. It is sufficient to just answer the questions with properly discussion, which will be the main emphasis. Do not write a traditional report. Source code should *not* be contained in the report.
- It is important to obtain the correct results and discuss them. If you use other sources than the text book or lecture notes, remember to *always* cite them.
- A well-documented and self-contained code satisfies the following criterions:
  - It includes sufficient information to make it clear for the user what the program does, and how to use it.
  - It executes and provides expected results without any problems. In particular this means that all submodules you write must be included in this file.
- There will be much emphasis on running time. When you are sure that the program is 100 % correct, examine whether the speed is optimal.

Nonlinear equations

$\boxed{1}$ **a)** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a multivariate scalar function, such that its gradient vector and Hessian matrix are respectively ∇f and Hf. Prove that if you want to find the zeros of ∇f by Newton iteration, then the scheme can be formulated as

$$\mathbf{x}_{m+1} = \mathbf{x}_m - Hf(\mathbf{x}_m)^{-1}\nabla f(\mathbf{x}_m) \quad , \quad \mathbf{x} \in \mathbb{R}^n \tag{1}$$

Create the program `Extremal_Points.py` with the following submodules:

- `XML_Extraction`, for reading XML-data and generating the functions.
- `Newton_Iteration`, for starting the iteration in (1).
- `Classify_Point`, for finding the eigenvalues of Hf at the stationary point.

The main program takes the name of the XML-file (string) and the initial guess (vector) as input arguments. The stopping criterion is $\|X_{m+1} - X_m\| \leq 10^{-14}$.

b) If $\mathbf{A}$ is a symmetric matrix, then its entries can be partitioned in the vectors $\mathbf{d}$ and $\mathbf{u}$ for the diagonal and upper triangular elements, respectively, such that

$$\mathbf{d} = [a_{11}, a_{22}, \ldots, a_{nn}]$$
$$\mathbf{u} = [a_{12}, \ldots, a_{1n}, a_{23}, \ldots, a_{2n}, \ldots, \ldots, a_{n-1,n}]$$

Write an algorithm (pseudocode) for assembling $\mathbf{A}$ by accessing the elements of $\mathbf{d}$ and $\mathbf{u}$ properly, with running time $\mathcal{O}(\frac{n^2+n}{2})$.
This can be used for generating the Hessian matrix from XML-data for any dimension n. You create a matrix of strings, and then convert everything into a *single string* with the same format as shown in the appendix (the auxiliary matrix cannot be converted directly with `str()`).

For the next remaining subtasks, the answers should include:

1. Explicit expressions for every partial derivative up to order 2.
2. The stationary points and their classification.

Use `Extremal_Points.py` for this purpose.

c) $f(x,y) = 2x^2y + 4xy - y^2$, three points in $[-2,0]^2$.

d) $f(x,y) = (x^2 - y^2)e^{-\frac{x^2+y^2}{2}}$, five points in $[-2,2]^2$.

e) $f(x,y) = (4x^2 + y^2)e^{-x^2-y^2}$, five points in $[-3,3]^2$.

f) $f(x,y,z) = xyz - x^2 - y^2 - z^2$, five points in $[-2,2]^3$.

Eigenvalues

$\boxed{2}$ **a)** Prove Gerschgorin's theorem about the localization of eigenvalues for matrices:

Let $n \geq 2$ and $\mathbf{A} \in \mathbb{C}^{n\times n}$. The eigenvalues of $\mathbf{A}$ lie in the region D, where

$$D = \bigcup_{i}^{n} D_i \quad , \quad D_i = \{z \in \mathbb{C} \, : \, |z - a_{ii}| \leq R_i\} \quad , \quad R_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$$

Create the program `Eigenvalue_Program.py` with the following submodules:

- `Matrix_Generator`, for creating the chosen matrix.
- `Run_Simulation`, for running one of the two iterations repeatedly.
- `Power_Eig`, for starting the power method.
- `QR_Eig`, for starting the QR-method.
- `Plot_Iterations`, for plotting the number of steps used in the iteration.

The program takes three arguments: two strings (program and algorithm indicator) and an integer (matrix indicator). The error tolerance should be 10^{-14}, and the initial vector can just be $(1/\sqrt{n}, \ldots, 1/\sqrt{n})$ for power iteration.
When running the simulations, the number of iterations should be stored in txt-files. Then you open these files and use them for plotting graphs. The chosen matrices are

- $\mathbf{A} = \text{tri}\{4, 11, 4\}$, $n = 100$, tridiagonal matrix.
- $\mathbf{B} = \text{penta}\{2 - 7, 20, -7, 2\}$, $n = 100$, pentadiagonal matrix.
- $\mathbf{C} = \text{hepta}\{6, -3, -7, 19, -7, -3, 6\}$, $n = 100$, heptadiagonal matrix.

The module for simulation takes an integer (matrix) and string (algorithm) as input. Use `Eigenvalue_Program.py` for the simulation and plotting. Give the result files appropriate names like Power_1.txt and QR_3.txt.

b) Test the power iteration method on $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ with $I = \{10, 11, 12, \ldots, 200\}$. Plot all the graphs in the same figure.

c) Test the QR-iteration method on $\mathbf{B}$ and $\mathbf{C}$ with $I = \{10, 11, 12, \ldots, 200\}$. Plot all the graphs in the same figure.

d) Discuss the results. Why did some matrices require fewer iterations than the other ones?

Interpolation

$\boxed{3}$ **a)** The Chebyshev points are defined on the interval $[-1, 1]$. Construct a formula for mapping them to an arbitrary interval $[a, b]$.

Create the program `Interpolation_Program.py` with the following submodules:

- `XML_Extraction`, for reading XML-data.
- `Partition`, for creating a uniform grid or Chebyshev grid on $I = [a, b]$.
- `Compute_Points`, for evaluating the interpolation polynomial at every point on I with resolution 0.01, and storing everything in a txt-file.
- `Lagrange_Newton_Coefficients`, returning the coefficients of the Lagrange interpolation polynomial in Newton form.
- `Lagrange_Newton_Evaluation`, for evaluating the Lagrange interpolation polynomial at a given point.
- `Hermite_Newton_Coefficients`, returning the coefficients of the Hermite interpolation polynomial in Newton form.
- `Hermite_Newton_Evaluation`, for evaluating the Lagrange interpolation polynomial at a given point.
- `Collect_Data`, looping through every txt-file in a given folder, storing data in arrays, and then returning them in a collected matrix.
- `Plot_Error`, plotting the discrete L^2 norm for each file.
- `Plot_Polynomial`, for plotting selected polynomials and the target function.

The main program takes four strings as arguments: XML-file, program (Evaluation, Error, Visualization), method (Lagrange, Hermite) and grid (Uniform or Chebyshev). The last argument is n, the polynomial degree (there are $n + 1$ points).
The evaluation points are in the interval $I = [a, b]$, with resolution 0.01, and the polynomial degree is between 2 and 20. Give the output files appropriate names like f1_Lagrange_Uniform_4.txt and f4_Hermite_Chebyshev_9.txt.
There are four functions to be used in `Interpolation_Program.py`:

$$f(x) = \frac{2x^2 - 1}{x^4 + 1} \qquad I = [-10, 10]$$

$$f(x) = \frac{\cos(\pi x)}{\cosh(x)} \qquad I = [-4, 4]$$

$$f(x) = \cos(2\pi x)e^{-x^2} \qquad I = [-2, 2]$$

$$f(x) = \frac{x}{|x|^3 + 1} \qquad I = [-10, 10]$$

Since there are *many* txt-files to handle in this task, it is best to store them in folders with almost the same name as the files themselves, excluding the number. Thus, you must switch between directories when collecting data from the txt-files. You will need 16 folders for storing all the files after the simulation (4 folders per function)

The discrete L^2-norm is given by

$$E = \sqrt{\frac{1}{N}\sum_{i=1}^{N} |f(x) - y_i|^2}$$

b) List up all the first-order derivatives of the functions to be interpolated.

c) Run the simulations and create 16 figures with one graph each for the discrete L^2-norm (semilog y) . Include these graphs in the report and describe their behaviour. Do you see any pattern?

d) Create 16 figures with five graphs each (the target function and the interpolation polynomials for $n = \{2, 4, 6, 8\}$). Include these graphs in the report and describe their behaviour. Do you see any pattern as you did previously? What can you conclude from all these simulations?

Appendix

XML-parsing

XML is used for storing data in separate files in order to reuse them efficiently without modifying the source code of the main program. In this way, you make the source code more general and generic. The main ingredient is the library `xml.etree.ElementTree`, which has been included with some other ones in the PYTHON-templates. The name of the XML-file for input can be anything, but the ending must always be `.xml`. If you use *XMLFILE* as a variable for the file name, you write the syntax

```
tree = et.parse(XMLFILE)
root = tree.getroot()
```

Thus, you have created a tree with a root, and elements are accessed like a linked list. All the XML-variables are strings by default. For example, if element number 3, 6 and 8 in the tree are respectively string, integer or float, you write

```
var3 = str(root[3].text)
var6 = int(root[6].text)
var8 = float(root[8].text)
```

The syntax `root[3].text` means that you go to the XML-tag with index 3. An XML-tag might have sub-tags, which again have more sub-tags. In that case, the syntax gets the form `root[3][2].text`, `root[2][5][2].text`, and so in.

If a variable is a scalar or vector function, you create a function handle by writing

```
f = lambda x : eval(x**2+np.cos(x)+1)
f = lambda x,y,z : eval(x*y**2-z*np.exp(-x**2))
F = lambda x,y : np.array(eval("[x**2+y**2-8,x**2-y**4]"))
```

The variables of a multivariate function can even be expressed as the entries of a vector:

```
f = lambda X : eval(X[0]+X[1]+X[2])
```

If the XML-file contains the entries of a 2×2-matrix function, then

```
h11 = root[0].text
h12 = root[1].text
h21 = root[2].text
h22 = root[3].text
H = "[["+h11+","+h12+"],["+h21+","+h22+"]]"
A = lambda x,y : np.array(eval(H))
```

When the function handle is defined, it can be taken as input or output from other functions in the main program.

General test for extremal points

If $f : \mathbb{R}^n \to \mathbb{R}$ is a multivariate scalar function with a stationary point $\mathbf{x}_0$, such that $\nabla f(\mathbf{x}_0) = \mathbf{0}$, then we have the following general test for classifying this point:

1. If every eigenvalue of $Hf(\mathbf{x}_0)$ is strictly positive, then $\mathbf{x}_0$ is a minimum.
2. If every eigenvalue of $Hf(\mathbf{x}_0)$ is strictly negative, then $\mathbf{x}_0$ is a maximum.
3. If $Hf(\mathbf{x}_0)$ has strictly positive and negative eigenvalues, then $\mathbf{x}_0$ is a saddle-point.
4. If $Hf(\mathbf{x}_0)$ has at least one zero eigenvalue, and all the others have the same sign, then $\mathbf{x}_0$ is unclassifiable.

Fast computing in Hermite interpolation

When evaluating a Lagrange interpolation polynomial at a given point, it is always best to use Newton's method of divided differences because it requires minimal computational effort, and the evaluation becomes stable. A similar method exists for evaluating Hermite interpolation polynomials too. We have a set of $n+1$ distinct numbers $\{x_i\}_{i=0}^{n}$, and create a new set of $2n+2$ numbers $\{z_i\}_{i=0}^{2n+1}$ such that

$$z_{2i} = z_{2i+1} \quad , \quad 0 \leq i \leq n$$

If we have three points, the scheme for divided differences becomes

z	$f(z)$	First divided differences	Second divided differences
$z_0 = x_0$	$f[z_0] = f(x_0)$	$f[z_0, z_1] = f'(x_0)$	$f[z_0, z_1, z_2] = \frac{f[z_1,z_2]-f[z_0,z_1]}{z_2-z_0}$
$z_1 = x_0$	$f[z_1] = f(x_0)$	$f[z_1, z_2] = \frac{f[z_2]-f[z_1]}{z_2-z_1}$	$f[z_1, z_2, z_3] = \frac{f[z_2,z_3]-f[z_1,z_2]}{z_3-z_1}$
$z_2 = x_1$	$f[z_2] = f(x_1)$	$f[z_2, z_3] = f'(x_0)$	$f[z_2, z_3, z_4] = \frac{f[z_3,z_4]-f[z_2,z_3]}{z_4-z_2}$
$z_3 = x_1$	$f[z_3] = f(x_1)$	$f[z_3, z_4] = \frac{f[z_4]-f[z_3]}{z_4-z_3}$	$f[z_3, z_4, z_5] = \frac{f[z_4,z_5]-f[z_3,z_4]}{z_5-z_3}$
$z_4 = x_2$	$f[z_4] = f(x_2)$	$f[z_4, z_5] = f'(x_0)$	
$z_5 = x_2$	$f[z_5] = f(x_2)$		

The Hermite interpolation polynomial gets the Newton form

$$H_{2n+1}(x) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, \ldots, z_k] \prod_{j=0}^{k-1} (x - z_j)$$

The QR algorithm for eigenvalues

The QR factorization method can be used for computing the whole spectrum of a matrix.

Algorithm 1 QR-algorithm for eigenvalues

```
1: procedure QR_EIGENVALUE($\mathbf{A}$)
2:     $n \leftarrow$ dimension of $\mathbf{A}$
3:     $L \leftarrow$ vector with $n$ zeros
4:     $N \leftarrow 0$
5:     $tol \leftarrow$ tolerance
6:     $\mathbf{A} \leftarrow$ Hessenberg($\mathbf{A}, n$)
7:     for i from n to 1 do
8:         $L_i, A, t \leftarrow$ QR_Shift($\mathbf{A}_{1:i,1:i}$,$i$,$tol$)
9:         $N \leftarrow N + t$
10:    return $L, N$
11:
12: procedure HESSENBERG($\mathbf{A}, n$)
13:    for k from 1 to n-2 do
14:        $\mathbf{z} \leftarrow \mathbf{A}_{k+1:n,k}$
15:        $\mathbf{e} \leftarrow (n-k)$-vector with zeros, and 1 at first entry
16:        $\mathbf{u} \leftarrow \mathbf{z} + (sgn(\mathbf{z}_1)\|\mathbf{z}\|_2)\mathbf{e}$
17:        $\mathbf{u} \leftarrow \mathbf{u}/\|\mathbf{u}\|_2$
18:        $\mathbf{A}_{k+1:n,k:n} \leftarrow \mathbf{A}_{k+1:n,k:n} - 2 \cdot \mathbf{u}(\mathbf{u}^T\mathbf{A}_{k+1:n,k:n})$
19:        $\mathbf{A}_{1:n,k+1:n} \leftarrow \mathbf{A}_{1:n,k+1:n} - 2 \cdot (\mathbf{A}_{1:n,k+1:n}\mathbf{u})\mathbf{u}^T$
20:    return $A$
21:
22: procedure QR_SHIFT($\mathbf{A}, m, tol$)
23:    $\lambda \leftarrow \mathbf{A}_{mm}$
24:    $t \leftarrow 0$
25:    $e \leftarrow 1$
26:    $\mathbf{I} \leftarrow$ identity matrix of size $m$
27:    if $m > 1$ then
28:        while $e > tol$ do
29:            $t \leftarrow t + 1$
30:            $\mathbf{Q}, \mathbf{R} \leftarrow$ QR($\mathbf{A} - \lambda\mathbf{I}$)
31:            $\mathbf{A} = \mathbf{RQ} + \lambda\mathbf{I}$
32:            $\lambda \leftarrow \mathbf{A}_{mm}$
33:            $e \leftarrow \mathbf{A}_{m,m-1}$
34:    return $\lambda$,$\mathbf{A}$,$t$
```

Special Python codes

An empty vector **a** or matrix **A** of strings can be initialized by

```
a = ["0"]*n
A = ["0"]*n
for i in range(0,n):
    A = ["0"]*m
```

The linear system $\mathbf{Ax} = \mathbf{b}$ is solved by writing

```
x = numpy.linalg.solve(A,b)
```

Calculation of the norm of a vector **x** requires correct option:

```
numpy.linalg.norm(x,1)                      # 1-norm
numpy.linalg.norm(x)                        # 2-norm
numpy.linalg.norm(x,numpy.inf)              # inf-norm
```

Special matrices:

```
A = numpy.zeros((4,4))
A = scipy.sparse.eye(7)
A = scipy.sparse.diags([-1,2,-1],[-1,0,1],shape=[n,n],format='lil')
A = A.todense()
```

The eigenvalues of **A** and its QR-decomposition are found by

```
S = scipy.linalg.eigvalsh(A)
Q,R = scipy.linalg.qr(A)
```

Syntax for the operations $\mathbf{Ab}$, $\mathbf{AB}$, $\mathbf{u}^T\mathbf{v}$ and $\mathbf{uv}^T$ is

```
numpy.dot(A,b)                              # Matrix-Vector product
numpy.dot(A,B)                              # Matrix-Matrix product
numpy.dot(u,v)                              # Vector dot product
numpy.outer(u,v)                            # Vector tensor product
```

If you need the name of a file, excluding the ending, you write

```
prefix = FILE.split('.')[0]
```

If the name of a folder is stored as a string, PATH, you can enter and exit by using

```
os.chdir(PATH)
os.chdir("..")
```

Useful functions in the process of reading and writing files:

```
open, close, writelines
```

Useful functions in the process of graph plotting:

```
subplots, plot, semilogy, set_xticks, set_yticks, set_xlim, set_ylim,
set_xlabel, set_ylabel, legend, savefig
```