Norwegian University of Science
and Technology
Department of Mathematical
Sciences

TMA4215 Numerical mathematics Autumn 2017

Project 2

This project counts for 20% of the final grade in the course.
The deadline is **November 24, 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 number(s), not your name(s).

Some hints:

- Before starting, make sure that you remember what you did in the previous project. The second exercise *must* be done in Python. We recommend the following resources:
  - https://www.python.org
  - https://docs.scipy.org/doc/
  - http://matplotlib.org
  - https://github.com/sintefmath/Splipy
  - http://www.pythonforbeginners.com/files/reading-and-writing-files-in-python
- 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, which is strongly recommended, 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.

Gauss-Legendre quadrature

1 **a)** (Challenge) Prove that Legendre polynomials satisfy the orthogonality relation

$$\int_{-1}^{1} L_n(x)L_m(x)\,dx = \delta_{mn}\frac{2}{2n+1}$$

For $m < n$, apply $(x^2-1)^n = (x-1)^n(x+1)^n$ and integration by parts. For $m = n$, use the substitution $x = \sqrt{y}$. Some relevant formulas:

Rodrigues' formula:	$L_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}[(x^2-1)^n]$
Beta integral:	$\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}\,dx = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$
Gamma identities:	$\Gamma(x+1) = x\Gamma(x)$, $\Gamma(n+1) = n!$

b) The Legendre polynomials are defined by the recursive relation

$$L_0(x) = 1 \quad , \quad L_1(x) = x$$
$$(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x) \qquad n \geq 1$$

Describe an algorithm with *linear* running time used for evaluating $L_n(x)$ at $x \in [-1,1]$ for $n \geq 0$. Explain how to use it for evaluating $L_n'(x)$ and $L_n''(x)$.

We will use Olver's method (project 1) for finding the zeros of $L_n(x)$, so we need a suitable initial guess. If $\{\cos(\theta_{kn})\}_{k=1}^n$ are zeros of $L_n(x)$, then

$$\frac{(2k-1)\pi}{2n+1} < \theta_{kn} < \frac{2k\pi}{2n+1} \quad , \quad 1 \leq k \leq n$$

Use this to derive an initial guess. Explain two reasons why it will always work.

c) Create the module `Gauss_Legendre`, which returns the numerical Gauss-Legendre quadrature and the analytical integral. The submodules are

- `XML_extraction`, extracting data from XML-files.
- `Gauss_Legendre_Data`, returning an $n \times 2$-matrix with nodes and weights.
  - `Olver`, finding the root of $L_n(x)$.
  - `Legendre_0`, evaluation of $L_n(x)$ (special treatment of $x = \pm 1$).
  - `Legendre_1`, evaluation of $L_n'(x)$ (special treatment of $x = \pm 1$).
  - `Legendre_2`, evaluation of $L_n''(x)$ (special treatment of $x = \pm 1$).
- `Gauss_Legendre_Quadrature`, performing the quadrature.
- `Return_Quadrature`, returning the numerical and analytical integrals. The three previous submodules are used in it.

The weights are given by the formula

$$\omega_i = \frac{2}{(1-x_i^2)[L_n'(x_i)]^2} \quad , \quad 1 \leq i \leq n$$

Explain how the zeros' properties can be used for *halving* the running time.

d) Consider the following functions to be integrated on $[-1, 1]$:

$$
\begin{aligned}
f(x) &= x^6 - 5x^4 + 7x^2 + 1 \\
f(x) &= x^{12} - 3x^8 + 6x^6 + 8x^2 - 3 \\
f(x) &= x^{18} - 8x^{14} - 2x^{12} + 10x^8 + 5x^4 - 3x^2 + 1 \\
f(x) &= x^4 \cos(8x) + x^2 \cos(4x) \\
f(x) &= x^2 e^{-3x} + x^3 e^{-2x} \\
f(x) &= x^3 \sin(2x) e^{-x}
\end{aligned}
$$

Create the main program, `Quadrature_Analysis`, using the previous module. It takes the following input parameters:

- XMLFILE, the name of the XML-file.
- PROGRAM, integer variable for choosing between repeated quadrature (1) or graph plotting (2).
- n_1 and n_2, number of integration nodes.

Create a for-loop in $I = [n_1, n_2]$, where you call `Return_Quadrature` from `Gauss_Legendre`, and return the numerical and exact integrals. The relative errors must be stored in an .asc-file. Open the .asc-file and plot the graph in .png-format. Thus, this program requires two different submodules:

- `Repeated_Quadrature`, performing repeated quadrature in a for-loop.
- `Convergence_Graph`, opening the .asc-file and plotting the graph.

Choose $I = [1, 10]$ and $I = [1, 20]$ for the three first and three last integrals. List the analytical integral values. Discuss and explain the behaviour of the graphs.

e) Explain how to apply Gauss-Legendre quadrature for the general finite integral

$$\int_a^b f(x)\, dx$$

You do not need to program, just give a complete explanation.

f) Most orthogonal polynomials have one common property. It is possible to find their zeros by finding the eigenvalues of a tridiagonal matrix whose entries have an analytical formula. Unfortunately, this method is not efficient when the number of zeros to find becomes large. Explain four reasons for this defect.

B-Spline quadrature

B-splines, which are piecewise defined and globally differentiable functions, can be defined by knot vectors, a nondecreasing sequence of knots given by

$$\boldsymbol{\tau} = \{\tau_1, \tau_2, \ldots, \tau_{n+p+1}\}$$

The B-splines are defined recursively by the Cox-de Boor formula:

$$N_{i,p}(\xi) = \frac{\xi - \tau_i}{\tau_{i+p} - \tau_i} N_{i,p-1}(\xi) + \frac{\tau_{i+p+1} - \xi}{\tau_{i+p+1} - \tau_{i+1}} N_{i+1,p-1}(\xi) \tag{1a}$$

$$N_{i,0}(\xi) = \chi_{[\tau_i, \tau_{i+1})} = \begin{cases} 1 & \tau_i \leq \xi < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{1b}$$

where $0/0 := 0$. We also have the derivative and integral formulas

$$\frac{d}{d\xi} N_{i,p}(\xi) = \frac{p}{\tau_{i+p} - \tau_i} N_{i,p-1}(\xi) - \frac{p}{\tau_{i+p+1} - \tau_{i+1}} N_{i+1,p-1}(\xi) \tag{2}$$

$$\int_{\mathbb{R}} N_{i,p}(\xi)\, d\xi = \frac{\tau_{i+p+1} - \tau_i}{p+1} \tag{3}$$

The spline function itself is a linear sum of B-splines:

$$\varphi(x) = \sum_{i=1}^{n} c_i N_i(x)$$

Thus, we can define the univariate space of spline functions as

$$\mathbb{S}^p(\boldsymbol{\tau}) = \left\{ \varphi \in L^2 \,:\, \phi(\tau_i) \in C^{k_i},\, \varphi|_{\xi \in (\tau_i, \tau_{i+1})} \in \mathbb{P}^p \right\}$$

If the continuity is r on each knot, we just write $\mathbb{S}^p_r$. We will focus on open knot vectors, where the first and last knot are repeated $p+1$ times. Exact integration yields

$$\begin{aligned} &\int_{\mathbb{R}} \varphi(\xi)\, d\xi = \sum_{i=1}^{n} w_i \varphi(\xi_i), \quad \forall \varphi \in \mathbb{S}^p(\Xi) \\ \implies &F_j\left(\begin{bmatrix} \mathbf{w} \\ \boldsymbol{\xi} \end{bmatrix}\right) = \sum_{i=1}^{n} w_i N_{j,p}(\xi_i) - \int_{\mathbb{R}} N_{j,p}(\xi)\, d\xi, \quad 1 \leq j \leq 2n \end{aligned}$$

Here, $\mathbf{F} \in \mathbb{R}^{2n}$ is the nonlinear vector function to be solved in order to find the weights and nodes for quadrature on the given knot vector.

2 **a)** Create the function `Basis_Plot.py`. Plot basis functions for the knot vectors

$$\boldsymbol{\tau} = [0,0,0,1,2,3,4,4,4]$$
$$\boldsymbol{\tau} = [0,0,0,1,1,2,2,3,3,4,4,4]$$
$$\boldsymbol{\tau} = [0,0,0,0,1,2,3,4,4,4,4]$$
$$\boldsymbol{\tau} = [0,0,0,0,1,1,2,2,3,3,4,4,4,4]$$

Relevant splipy-functions: `BSplineBasis`, `evaluate` (the order is $p+1$, where p is the polynomial degree of the spline).

b) Express $\frac{\partial \mathbf{F}}{\partial \mathbf{w}}$ and $\frac{\partial \mathbf{F}}{\partial \boldsymbol{\xi}}$ as $\mathbb{R}^{2n\times n}$-matrices, and rewrite $\mathbf{F}$ on the form $\mathbf{Ab}+\mathbf{c}$. Use this to derive the scheme

$$\begin{bmatrix}\mathbf{w}\\ \boldsymbol{\xi}\end{bmatrix}_{n+1} = \begin{bmatrix}\mathbf{w}\\ \boldsymbol{\xi}\end{bmatrix}_{n} - \partial\mathbf{F}_n^{-1}\mathbf{F}_n \qquad , \qquad \mathbf{F}_n = \mathbf{F}\left(\begin{bmatrix}\mathbf{w}\\ \boldsymbol{\xi}\end{bmatrix}_{n}\right)$$

c) By applying that $\frac{\partial \mathbf{F}}{\partial \mathbf{w}}$ and $\frac{\partial \mathbf{F}}{\partial \boldsymbol{\xi}}$ have a band structure, propose a method for reducing the bandwidth of the Jacobian

$$\partial\mathbf{F} = \left[\frac{\partial \mathbf{F}}{\partial \mathbf{w}}, \frac{\partial \mathbf{F}}{\partial \boldsymbol{\xi}}\right] \in \mathbb{R}^{2n\times 2n} \tag{4}$$

d) Write the script `Spline_Quadrature`, using Newton iteration to find nodes and weights for the knot vectors in **2a)**. This program has three parts:

- `Prepare_Data`, for creating the initial condition and augmenting the knot vector. It must also return n and the constant integral vector in $\mathbf{F}$.
- `Assembly`, for updating $\mathbf{F}_n$ and $\partial\mathbf{F}_n$ every time in the Newton iteration. Recall that $\partial\mathbf{F}_n$ must be permuted and made sparse.
- `Spline_Quadrature`, the main program with Newton iteration.

If the number of basis functions, $2n$, is odd, you can augment the knot vector by adding an extra knot $(\tau_{p+1}+\tau_{p+2})/2$ between τ_{p+1} and τ_{p+2}. For this task, use 10^{-11} as the error tolerance.
Relevant SciPy-functions: `sparse.csr_matrix`, `sparse.linalg.spsolve`. The arrays obtained after using `evaluate` must be transposed.

e) Create the function `Curve_Plot.py`. Use the file *Curve.g2* in *Code facilities* and plot $[x(t), y(t)]^T$. The process of reading g2-files with splipy is not so straightforward. It is important to check the instruction manual first.
Relevant functions: `G2`, `Read`, `"with .. as"` (reserved syntax).

f) Create `Read.py`, using the quadrature algorithm for calculating $\int_L ds$. Recall that ds is the differential describing the path.
Relevant functions: `knots`, `order`, `derivative`.

g) Create the function `Area_Plot.py`. Use the file *Area.g2* in *Code facilities* and plot the area.

h) Calculate the area $\int_A dA$ from above. You need the Jacobian's determinant. It can be found by cross-product of derivatives (`cross(du,dv)` from NumPy works). For the surface object, `knots`, `order` and `derivative` are available.