(b) Algorithm Implementation Write a computer code to solve the discretised problem with the indicated method.

• Provide a print-out of your code in your report.

(c) Code Verification Carry out a verification of your implementation by comparing computed approximations with the indicated exact solution:

• Plot a sufficiently-coarse (but not too coarse) approximation and the exact solution in one figure. (The difference between both should be visible.) In case of a 2D domain, plot the approximation and solution versus x for $y = 0.25$, 0.5 and 0.75. In case of a time-dependent problem, plot the approximation and solution versus x for $t = 0$, $T/2$ and T.

• Study the convergence of the method: Define an appropriate norm. Provide a table with the norm of errors for a sequence of appropriate mesh-sizes (and corresponding time-step sizes in case of a time-dependent PDE). Plot the errors versus the mesh-widths (using double-log axes). Comment on the observed rates of convergence.

Choice FML (Finance and Machine Learning choice)

FML-1 **1-D Elliptic PDE** (See Lecture 6) **[15 marks]**

First consider the 1D elliptic PDE for $u : (0,1) \to \mathbb{R}$ subject to Dirichlet BCs (boundary conditions):

$$
\begin{aligned}
-u'' &= f \qquad \text{for } x \in (0,1) && \text{(1a)}\\
u(0) &= \alpha && \text{(1b)}\\
u(1) &= \beta && \text{(1c)}
\end{aligned}
$$

with $f : (0,1) \to \mathbb{R}$, $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$.

(a) **PDE Discretisation** [See **Q-0(a)**] Obtain the discretisation for a second-order difference scheme as explained in Chapter 2 in [LeVeque, 2007].

(b) **Algorithm Implementation** [See **Q-0(b)**]

(c) **Code Verification** [See **Q-0(c)**] Verify against the exact solution

$$u_{\text{exact}}(x) = x - \sin(\pi x)\,, \tag{2}$$

using the data:[1]

	α	β	$f(x)$
Case 1:	0	1	$-u''_{\text{exact}}(x)$

[1] The verification technique, where a given exact solution is used to generate the data in the problem, is referred to as the *manufactured-solution technique*.

FML-2 **Heat Equation** (See Lecture 7) **[15 marks]**

Consider the parabolic PDE (heat equation) for $u(t,x) \in \mathbb{R}$ subject to Dirichlet BCs and an initial condition:

$$u_t - a u_{xx} = 0 \qquad \text{for } t \in (0,T]\,,\; x \in (0,1) \tag{3a}$$

$$u(t,0) = g_0 \tag{3b}$$

$$u(t,1) = g_1 \tag{3c}$$

$$u(0,x) = u_0(x) \tag{3d}$$

with $a > 0$ (heat-conduction coefficient), $T > 0$ and $u_0 : (0,1) \to \mathbb{R}$.

The solution $u(t,x)$ represents the temperature, at time t and position x, in a one-dimensional piece of conductive material.

(a) PDE Discretisation [See **Q-0(a)**] Obtain the discretisation for the implicit method (backward difference in time, centred in space) as explained in Chapter 9 in [Epperson, 2013].

(b) Algorithm Implementation [See **Q-0(b)**]

(c) Code Verification [See **Q-0(c)**] Verify against the exact solution

$$u_{\text{exact}}(t,x) = \mathrm{e}^{-4t}\sin(2\pi x) + x\,, \tag{4}$$

using the data:

	a	T	g_0	g_1	$u_0(x)$
Case 2:	π^{-2}	1	0	1	$\sin(2\pi x) + x$

Hint: In studying the convergence, take $\Delta t = Ch^2$ for some chosen C (with h the mesh width and Δt the time-step size). Measure errors in the max-norm at the final time T, in other words,

$$\max_i |e_i^N|\,, \tag{5}$$

with N such that $N\Delta t = T$.

FML-3 **Black–Scholes Equation (European option)** (Cf. Lecture 7, 8) **[20 marks]**

Consider the parabolic PDE (Black–Scholes equation) for $v(t,x) \in \mathbb{R}$ subject to time-dependent Dirichlet BCs and an initial condition:

$$\frac{\partial v}{\partial t} - \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2} - r\,x\,\frac{\partial v}{\partial x} + r\,v = 0 \qquad \text{for } t \in (0,T]\,,\ x \in (0,R) \tag{6a}$$

$$v(t,0) = f_0(t) \tag{6b}$$

$$v(t,R) = f_R(t) \tag{6c}$$

$$v(0,x) = g(x) \tag{6d}$$

with $\sigma > 0$ (constant volatility), $r > 0$ (interest rate), $T > 0$, $R > 0$, $f_0 : (0,T] \to \mathbb{R}$, $f_R : (0,T] \to \mathbb{R}$, and $g : \mathbb{R} \to \mathbb{R}$ (pay-off function).

The solution $v(t,x)$ represents the value of a European option with maturity time T, at time-to-maturity t and spot price x.[2]

(a⋆) PDE Discretisation [See **Q-0(a)**] Use the following *implicit* finite-difference scheme for (6a)–(6d): Consider the PDE (6a) at an arbitrary point $(x,t) = (x_i, t_{n+1})$, then replace $\frac{\partial v}{\partial t}$ by $\frac{v_i^{n+1} - v_i^n}{\Delta t}$, replace $\frac{\partial^2 v}{\partial x^2}$ by the standard second-order central difference approximation at t_{n+1}, replace $\frac{\partial v}{\partial x}$ by $\frac{v_{i+1}^{n+1} - v_i^{n+1}}{h}$, and replace v by v_i^{n+1}.

(b⋆) Algorithm Implementation [See **Q-0(b)**]

(c⋆) Code Verification I [See **Q-0(c)**] Verify against the exact solution

$$v_{\text{exact}}(t,x) = K\mathrm{e}^{-rt}\,\Phi\big(-d_-(t,x)\big) - x\,\Phi\big(-d_+(t,x)\big)\,, \tag{10}$$

where

$$d_\pm(t,x) := \frac{1}{\sigma\sqrt{t}}\Big(\ln\big(x/K\big) + \big(r \pm \tfrac{\sigma^2}{2}\big)t\Big) \tag{11}$$

and $\Phi(\cdot)$ is the cumulative distribution function[3] of the standard normal distribution:

$$\Phi(d) := \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{d} e^{-z^2/2}\,\mathrm{d}z\,, \tag{12}$$

using the data:

[2] In particular, v is related to V by $v(T-t,x) = V(t,x)$, where $V(t,x)$ is the value of the European option in *forward* time t, and which satisfies

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 V}{\partial x^2} + r\,x\,\frac{\partial V}{\partial x} - r\,V = 0\,, \tag{7}$$

subject to the *terminal* condition $V(T,x) = g(x)$. The value of the European option is given by the (discounted) expected pay-off at maturity:

$$V(t,x) = \mathrm{e}^{-r(T-t)}\,\mathbb{E}\Big[g\big(S(T)\big)\,\Big|\,S(t) = x\Big]\,, \tag{8}$$

where $S(t) = S_0\,\mathrm{e}^{(r-\sigma^2/2)t + \sigma W(t)}$ is the stock price according to the stochastic differential equation:

$$\mathrm{d}S = rS\,\mathrm{d}t + \sigma S\,\mathrm{d}W\,. \tag{9}$$

The connection between (8) and (7) is given by the Feynman–Kac formula.

[3] It is recommended to use a built-in function for this Gaussian cumulative distribution function. For example,

	r	σ	T	R	$f_0(t)$	$f_R(t)$	K
Case 3(i):	0	0.5	5	300	$K\,\mathrm{e}^{-rt}$	$v_{\text{exact}}(t,R)$	100

and[4]

$$g(x) = \begin{cases} K - x & \text{if } x < K\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{13}$$

(d⋆) Code Verification II Repeat Question **(c)** but now for the data:

	r	σ	T	R	$f_0(t)$	$f_R(t)$	K
Case 3(ii):	0.1	0.1	5	300	$K\,\mathrm{e}^{-rt}$	$v_{\text{exact}}(t,R)$	100

$\Phi(x)$ is related to built-in functions `erf` and `erfc`:

$$\Phi(x) = \frac{1}{2}\texttt{erfc}\big(-\frac{x}{\sqrt{2}}\big) = \frac{1}{2}\Big(1 - \texttt{erf}\big(-\frac{x}{\sqrt{2}}\big)\Big)$$

[4]The pay-off in (13) is typical for a *put* option with strike price K.

FML-4 **Gradient Methods** (See Lecture 5) **[25 marks]**

Consider the 1D-input-1D-ouput classification problem, where the goal is to construct the map $F : (0, 1) \to \mathbb{R}$ that minimizes

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\frac{1}{2}\Big(y(x_i) - F(x_i)\Big)^2}_{= C_{x_i}} \tag{14}$$

where $y(x_i)$ is a given output for the input data x_i, $i = 1, \ldots, N$. For this Problem Set, let $F(\cdot)$ be given by a simple linear polynomial, i.e.,

$$F(x) = w\,x + b\,,$$

where the parameters w and b are to be trained.

(a) **Algorithm Implementation I** Write a computer code that implements for this problem the *Gradient Descent Method* with learning-rate parameter η, initial parameter guess $\underline{\boldsymbol{p}}^{(0)} = (w^{(0)}, b^{(0)})$, and maximum number of iterations MaxIter.
- Provide a print-out of your code in your report.

(b) **Code Verification I** Consider the following data

	N	η	$w^{(0)}$	$b^{(0)}$	MaxIter
Case 4	3	0.75	0.5	0.5	64

i:	1	2	3
x_i:	0	0.5	1
$y(x_i)$:	0	1	1

$\Big($For this simple F, one can compute the (LSQ) minimizer: $(w_{\min}, b_{\min}) = (1, \frac{1}{6})$.$\Big)$
- Generate a table with the first 16 values of the parameters $(w^{(j)}, b^{(j)})$ and the first 16 values of the corresponding Cost.
- Plot in a figure the Cost versus the iteration number $j = 0, 1, 2, \ldots,$ MaxIter.

(c) **Algorithm Implementation II** Write now a computer code that implements the *Stochastic Gradient Method* with learning-rate parameter η, initial parameter guess $\underline{\boldsymbol{p}}^{(0)} = (w^{(0)}, b^{(0)})$, and maximum number of iterations MaxIter.
- Provide a print-out of your code in your report.

(d) **Code Verification II** Consider the same data as before, but do some numerical experiments yourself to find proper values for η and MaxIter.
- Which values work well?
- Generate a table with the first 16 values of the parameters $(w^{(j)}, b^{(j)})$ and the first 16 values of the corresponding Cost.
- Plot in a figure the Cost versus the iteration number $j = 0, 1, 2, \ldots,$ MaxIter.

FML-5 **Deep Learning Problem** (See Lecture 5) **[25 marks]**

Consider finally the problem discussed at the beginning of Section 2 and in Section 6 of [Higham, Higham, 2019]. This is a 2D-input-2D-output classification problem with map $\underline{\boldsymbol{F}} : (0,1) \times (0,1) \rightarrow \mathbb{R}^2$ given by an Artificial Neural Network, which minimizes

$$\text{Cost} = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\frac{1}{2} \left\| \underline{\boldsymbol{y}}(\underline{\boldsymbol{x}}^{\{i\}}) - \underline{\boldsymbol{F}}(\underline{\boldsymbol{x}}^{\{i\}}) \right\|^2}_{= C_{\underline{\boldsymbol{x}}^{\{i\}}}} \tag{15}$$

Note that the data is given by:

i:	1	2	3	4	5	6	7	8	9	10
$x_1^{\{i\}}$:	0.1	0.3	0.1	0.6	0.4	0.6	0.5	0.9	0.4	0.7
$x_2^{\{i\}}$:	0.1	0.4	0.5	0.9	0.2	0.3	0.6	0.2	0.4	0.6
$y_1(\underline{\boldsymbol{x}}^{\{i\}})$:	1	1	1	1	1	0	0	0	0	0
$y_2(\underline{\boldsymbol{x}}^{\{i\}})$:	0	0	0	0	0	1	1	1	1	1

(a$\star$) Algorithm Implementation

- **If not using Matlab:** Write code that implements the same neural network and the same stochastic gradient method with backpropagation step, as described in Section 6 of [Higham, Higham, 2019]. In other words, you can simply convert the Matlab code in Section 6 into your own language. Provide a print-out of your code in your report.
- **If using Matlab:** Since Section 6 of [Higham, Higham, 2019] already has working Matlab code, you are asked to extend that code by implementing a generalization of the neural network. For example, your network has more layers and/or more units per layer. Provide a print-out of your code in your report.

(b$\star$) Code Verification Come up with a verification of your code, and give a brief description. Mention all the parameters that you have set. Think about important outputs such as a plot of the Cost versus iterations during training, and the classification of example input values, etc.