First Name: **Signature:**

Last Name:

SCIPER ID: _____

EPFL

Final exam SOLUTION

CS328 - Numerical Methods in Visual Computing

Procedures:

This is a timed examination. You will have a maximum of **2 hours** to complete your answers. Leave your **student ID card** on the table during the entire time so that it can be verified. Write your **first and last name** as well as your **EPFL SCIPER number** on the cover page as well as every additional sheet of paper that you request during the exam. Use only black or blue ink pens and write legibly. Exam questions done in pencil or other colored pens will **NOT** be graded. Write your solutions into the space below answers and indicate if an answer is continued elsewhere.

Multiple-choice items are each worth one point, while incorrect answers deduct one point. The final score of multiple-choice question is clamped to a non-negative number. Please try to follow Python syntax when answering questions that ask for code. Minor syntax errors are tolerated.

Regulations:

With the exception of one (1) double-sided A4 page of personal notes, any use of textbooks or other books/printed materials, formula sheets, electronic devices such as calculators, laptops, cell phones or other PDAs, MP3 players, headphones, etc. is strictly **PROHIBITED** during the examination.

A student is deemed to have failed the course if he or she is caught cheating or found to be in violation of the above regulations.

Exercise	*Max. points*	*Earned points*
1. FP Arithmetic and Interpolation	30	
2. Linear Least Squares	15	
3. The Pseudoinverse and Conditioning	25	
4. Optimization	30	
Total	**100**	

Course Name:	Numerical Methods in Visual Computing	**Date:**	27.01.2023
Course Number:	CS 328	**Time:**	15:15-17:15
Lecturer:	Wenzel Jakob		

1 FP Arithmetic and Interpolation (30 pts)

1.1 Arithmetic, Part 1 (8 pts)

The following functions repeatedly apply one of the basic arithmetic operations:

```
def add_it(x):
    for i in range(10):
        x += x
    return x
```

```
def mul_it(x):
    for i in range(10):
        x *= x
    return x
```

```
def div_it(x):
    for i in range(10):
        x /= x
    return x
```

For each of the following `print` statements, state whether the operation returns

(a) an ordinary nonzero number,
(b) ± 0,
(c) $\pm\infty$,
(d) NaN (Not a Number),
(e) or does not terminate.

You don't need to provide the actual result value (just the case (a)-(e) is enough). The type `float32` implements the IEEE 754 specification including handling of special values.

```
from numpy import float32 as f32

print(add_it(f32(2))) # 1
print(mul_it(f32(2))) # 2
print(div_it(f32(2))) # 3
print(div_it(f32(0))) # 4
```

Solution:

1. a (2048.0)
2. c
3. a (1.0)
4. d

1.2 Arithmetic, Part 2 (8 pts)

We now map each of the values through the function `zero_it()` provided below. Once more, classify each of the `print` statements according to the earlier list.

```
def zero_it(x):
    it = 0
    while x > 0:
        x /= 2
        it += 1
    return it

print(zero_it(add_it(f32(2)))) # 1
print(zero_it(mul_it(f32(2)))) # 2
print(zero_it(div_it(f32(2)))) # 3
print(zero_it(div_it(f32(0)))) # 4
```

Solution:

1. a
2. e
3. a
4. b

1.3 Interpolation (10pts)

You are given a dataset consisting of 1000 points (x_i, y_i), where $x_i = i$. Assuming that computation time does not matter, list two severe issues that you might encounter when fitting a degree-999 polynomial to this data. For each issue, explain whether this issue is avoidable (and if so, how).

Solution:

- The Vandermonde matrix is ill-conditioned, and the solution explodes. This is avoidable: conditioning problems can be avoided by expressing the linear system using orthogonal polynomials.
- Runge's phenomenon will lead to a very badly behaved oscillatory interpolant due to the high degree and regular discretization. This is unavoidable.

Wrong answers: anything involving least squares, condition-squaring effects, LU vs QR factorization

Besides addressing the above two issues, what potentially desirable property would a cubic spline interpolant have?

Solution: The influence of each data point would be localized, which avoids global effects of problematic outliers. It also increases efficiency, since fewer coefficients must be loaded and referenced in computation to evaluate the fit at a given position. (Only mentioning one of these two properties is fine)

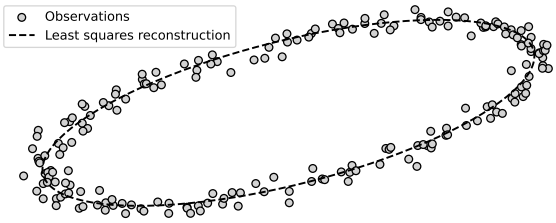
1.4 Multiple choice (4 pts)

True	False	
☐	☐	When two floating point values `a` and `b` are almost equal, their subtraction `a-b` suffers from catastrophic cancellation, causing it to be performed less accurately than the usual error guarantees of the IEEE-754 standard.
☐	☐	Any linear combination of the first N monomials (1, x, x^2, ..) can also be expressed as a linear combination of the first N Legendre polynomials .
☐	☐	The result of floating point operations is always deterministic.
☐	☐	A change in the ULP (unit in the last place) for the value $3x$ is three times as large as that for x.

Solution: B and C

2 Linear Least Squares (15 pts)

You are given n data points (x_i, y_i) $(i = 1, \ldots, n)$ on an ellipse that are, however, contaminated by a significant amount of noise:



Your task is to reconstruct the parameters of this ellipse using the least squares method. An ellipse can be parameterized by constants a, b, c, d, e so that the following equation holds for positions (x, y) on the boundary:

$$ax^2 + by^2 + cxy + dx + ey - 1 = 0$$

2.1 Least Squares Fit (10 pts)

Write down the least squares system that describes this problem. You don't need to factorize the system or apply the normal equations.

Solution:

$$\begin{bmatrix} x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 \\ \vdots & & & & \vdots \\ x_n^2 & y_n^2 & x_n y_n & x_n & y_n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

2.2 Variable Noise Characteristics (5 pts)

Later, you realize that some points are much noisier than others. Luckily, the scale of the noise is predictable, and you therefore decide to perform a *weighted* least squares reconstruction that minimizes

$$E(a, b, c, d, e) = \sum_{i=1}^{N} (w_i (a x_i^2 + b y_i^2 + c x_i y_i + d x_i + e y_i - 1))^2$$

Specify the least squares system for alternative problem assuming known weights w_i. As before, you don't need to factorize the system or apply the normal equations.

Solution:

$$\begin{bmatrix} w_1 x_1^2 & w_1 y_1^2 & w_1 x_1 y_1 & w_1 x_1 & w_1 y_1 \\ \vdots & & & & \vdots \\ w_n x_n^2 & w_n y_n^2 & w_n x_n y_n & w_n x_n & w_n y_n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

3 The Pseudoinverse and the Conditioning (25 pts)

In class, we had introduced the *pseudoinverse* $\mathbf{A}^+$ as a generalization of the inverse $\mathbf{A}^{-1}$. Compute the following matrix-vector products involving a pseudoinverse, and show intermediate steps or explain intuitively why the answer is correct.

Note: In expressions like $\mathbf{x} = \mathbf{A}^+\mathbf{b}$, it may be helpful to think of $\mathbf{x}$ as a particular solution of a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ that is potentially *singular*, *overdetermined*, or *underdetermined*.

Note 2: the pseudoinverse was introduced in the context of the Singular Value Decomposition (SVD), but an SVD is not needed below. You can find all answers by reasoning about the high-level properties of the pseudoinverse introduced in class.

3.1 Problem 1 (3 pts)

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & 0 \end{pmatrix}^+ \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

What is the value of x and y?

Solution: In the square case, the pseudoinverse works just like an ordinary inverse, but it leaves singular dimensions untouched. In this example, we have $x = 4$, $y = 0$.

3.2 Problem 2 (7 pts)

$$(x) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}^+ \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

What is the value of x?

Solution: In the tall case, the pseudoinverse gives the least squares solution to the linear system. In this case, it minimizes $(x-0)^2 + (x-1)^2 = 2x^2 - 2x + 1$. The minimum is achieved at $x = 1/2$.

A more geometric way to think about the problem is also fine (or even a drawing): the span of the matrix is a diagonal line passing through the origin with slope $(1, 1)$.
The solution can be found by perpendicularly projecting the vector $b = (0, 1)$ along the normal $n = (1, -1)$ of this line, i.e.: $b - n(n \cdot b)/(n \cdot n) = (0, 1) - (-1/2, 1/2) = (1/2, 1/2)$.

This position can be reached by traveling $x = 1/2$ along the line $(1, 1)$.

3.3 Problem 3 (7 pts)

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 \end{pmatrix}^{+} \begin{pmatrix} 1 \end{pmatrix}$$

What is the value of x and y?

Solution: In the wide case, there are many potential solutions, and the pseudoinverse provides the minimum-norm one. The linear system here states

$$\frac{1}{2}x + \frac{1}{2}y = 1$$

In other words, $x + y = 2$, which implies $y = 2 - x$. Then

$$\|(x, y)\|^2 = x^2 + (2 - x)^2 = 2x^2 - 4x + 4$$

The minimum is reached when $4x - 4 = 0$, which happens when $x = y = 1$.

3.4 Condition number (4 pts)

What is the condition number of the following matrix?

$$\mathbf{A} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/4 \end{pmatrix}$$

Solution: $\text{cond}(\mathbf{A}) = \|\mathbf{A}\|_2 \cdot \|\mathbf{A}^{-1}\|_2 = \frac{1}{2} \cdot 4 = 2$ (the answer alone is enough)

3.5 Multiple choice (4 pts)

An ill-conditioned linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is characterized by which of the following statements?

True	False	
☐	☐	All solutions of such a system will contain a significant amount of numerical error.
☐	☐	The singular value decomposition of $\mathbf{A}$ is not well-defined.
☐	☐	The smallest singular value of the matrix $\mathbf{A}$ is much smaller than 1.
☐	☐	A tiny relative perturbation of the right-hand side $\mathbf{b}$ can lead to a large perturbation of the solution $\mathbf{x}$.

Solution: Only D

4 Optimization (25 pts)

4.1 Backpropagation (10 pts)

In this exercise, we will compute the gradient of the function

$$\texttt{e}(\texttt{a},\texttt{b}) = \exp(\texttt{ab}) - \texttt{ab}.$$

The computer implementation of this function splits the expression into a sequence of smaller steps involving temporary values `c` and `d`.

```
a = ..        # 'a' and 'b' are
b = ..        # computed elsewhere
c = a * b
d = exp(c)
e = d - c
```

Implement a corresponding back-propagation (reverse-mode differentiation) pass that computes the derivatives `δa` and `δb` from a derivative `δe` of the output.

It may be helpful to first draw the graph structure of this computation, and annotate the sensitivity (derivative) of each node with respect to its predecessors along edges.

Note: This question is specifically about backpropagation. Other ways of computing the derivative (e.g. MATH-101 style) don't count.

Solution:

```
δe = ..
δd = δe
δc = d * δd - δe
δb = a * δc
δa = b * δc
```

Small ambiguities are possible, so don't grade this too strictly. That said, I'd want to see AD code that references primal variables.

```
δe = .. # δe is computed elsewhere
```

4.2 Multiple choice (5 pts)

True	False	
☐	☐	Due to the cheap gradient principle, memory usage of backpropagation will not be much bigger than that of the original program.
☐	☐	Backpropagation can compute derivatives with respect to multiple outputs of a function in an single pass.
☐	☐	Symbolic differentiation is generally more accurate than finite differences.
☐	☐	While gradient descent does not always converge to a global minimum, it is guaranteed to at least converge to a local minimum.
☐	☐	The main difference between automatic and MATH-101 style differentiation is the that automatic differentiation uses the chain rule.

Solution: True: only C

4.3 Bisection (10 pts)

Provide an implementation of the bisection algorithm that finds a root of a monotonic function `f(x)` on the interval `[l, r]`. Your code should *not* perform *unnecessary function evaluations* in every iteration.

Implement a loop that runs for a fixed number of 10 iterations (in other words: you don't need to provide a stopping criterion). The use of complex data structures like hash tables, caches, etc., is not allowed and will give 0 points.

Solution:

```
def bisect(f, l, r):
    m = (l + r) / 2                # Or 'l + (r - l) / 2'
    fl, fm = f(l), f(m)
    for i in range(10):
        if fl * fm < 0:
            r = m
        else:
            l = m
            fl = fm
        m = (l + r) / 2
        fm = f(m)
    return m
```

This exercise expects Python code as answer. Since the students don't have a computer to test their solutions, we should be quite lenient in grading. What I expect to see: up-front evaluation of some data points (2 or 3), and only a single function evaluation per iteration. There code should update the positions and cached values, though mixups here are forgivable.