

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 **3 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.

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.

<i>Exercise</i>	<i>Max. points</i>	<i>Earned points</i>
1. Floating point arithmetic	30	
2. Linear and least squares problems	40	
3. Neural networks and optimization	30	
Total	100	

Course Name: Numerical Methods in Visual Computing **Date:** 31.01.2018
Course Number: CS 328 **Time:** 8:15-11:15
Lecturer: Wenzel Jakob

1 Floating point arithmetic (30 pts)

1.1 Multiple choice (5 pts)

The following multiple-choice questions cover characteristic properties of floating point computations. You should assume that arithmetic and number representations use the IEEE 754 standard. Circle all items that are true.

- A. The identity $(x \leq 0 \mid \mid x > 0)$ holds for every x .
- B. There is an ϵ such that $(1 + \epsilon) + \epsilon = 1$, but $1 + (\epsilon + \epsilon) > 1$.
- C. The result of an elementary arithmetic operation involving two floating-point numbers is not always representable as a floating-point number.
- D. Given a finite number x , the identity $x - x - x == x * (-1.0)$ always holds.
- E. Nonsensical operations like subtracting infinity from infinity cause the application to terminate with an exception.

Solution: B, C, D

1.2 Fused multiply-add (7 pts)

Recent processors include an instruction `fma(a,b,c)` known as “fused multiply-add”, which efficiently computes the expression $a \cdot b + c$ in a single step—in other words: faster than doing a multiplication followed by an addition. The accuracy guarantees of this combined operation match those of other elementary arithmetic operations like addition or multiplication.

- (i) After upgrading a program to use FMA instructions, you notice that the following line of code in a program causes problems. When $x = y$, it sometimes returns a Not-a-Number (“NaN”) result.

```
z = sqrt(fma(x, x, -y*y))
```

Before the upgrade, the following code worked perfectly:

```
z = sqrt(x*x - y*y)
```

Explain the cause of this problem (5 pts).

Solution: When $x = y$, both $x * x$ and $y * y$ rounded to the same value, in which case $x * x - y * y = 0$ and $\sqrt{0} = 0$.

After enabling FMA instructions, one multiplication is done as usual, and one is part of the combined FMA instruction. Therein lies the problem: IEEE 754 elementary arithmetic operations are internally performed in infinite precision, and $x * x$ may not longer be equal to the rounded $y * y$. In this case, the result of the FMA operation is the rounding error, which can be negative (and attempting to take the square root of a negative number results in an NaN).

- (ii) Most modern compilers include a special flag to automatically find all expressions of the form $a \cdot b + c$ in a program and convert them to `fma(a, b, c)`. However, this flag is usually not turned on by default. Explain why an automatic conversion may be undesirable. Assume that the problem discussed in the previous paragraph does not apply (i.e. don't write down the same reason twice). (2 pts)

Solution: Due to the subtle rounding-related differences between `fma(a, b, c)` and $a \cdot b + c$, a program will generate different results depending on whether this flag is enabled or not. This is very much in violation of the idea of IEEE 754, which is to have standardized arithmetic that leads to identical results everywhere.

1.3 Function approximation (10 pts, 5 each)

While reviewing code that is part of a numerical library, you find a function to approximate the exponential function using a truncated power series:

$$\exp(x) \approx \sum_{i=0}^n \frac{x^i}{i!}$$

However, the code is not a direct translation of the above formula. Instead, it reads

```
from scipy.misc import factorial

def exp_approx(x, n):
    if x < 0:
        return 1 / exp_approx(-x, n)
    else:
        return sum(sorted([ x**i/factorial(i) for i in range(n) ]))
```

Note that the function `sorted` takes a list as input and sorts it in order of increasing magnitude (for example, `sorted([3, 1, 2]) == [1, 2, 3]`).

- (i) What is the purpose of branch in the first line of the function body? What would happen if the `x < 0` special case was removed?

Solution: The branch applies the exponential identity $\exp(-x) = 1/\exp(x)$ to deal with negative inputs, which ensures that all series terms are positive—otherwise, we'd add positive and negative terms, causing cancellation.

- (ii) What is the purpose of the `sorted` function call? What would happen if the sorting order was reversed?

Solution: The `sorted` call ensures that we add small entries first before moving onto larger ones. If the sort order was reversed, the last iteration would add a tiny element into a large accumulator, which will potentially not change the accumulator value at all.

1.4 Matrix multiplication (8 pts)

Suppose that a computation must transform n 10-dimensional vectors $\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^{10}$ by two linear transformations represented using matrices $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{R}^{10 \times 10}$.

One way to do this requires two matrix multiplications per vector:

$$\mathbf{b}_i = \mathbf{A}_1(\mathbf{A}_2\mathbf{x}_i). \quad (i = 1, \dots, n)$$

Alternatively, we could compute $\mathbf{A}_3 = \mathbf{A}_1\mathbf{A}_2$ once and then use a single multiplication per vector:

$$\mathbf{b}_i = \mathbf{A}_3\mathbf{x}_i \quad (i = 1, \dots, n)$$

- (i) Compute the number of floating point operations (additions & multiplications) for methods 1 and 2. You are allowed to make small approximations – up to 10% error are tolerable. (5 pts)

Solution: A 10D dot product requires 9 mult., 10 add. – let's say 20 flops. Hence a matrix-vector product requires 200 flops, and a matrix-matrix multiplication requires 2000. Method 1 thus requires $400 \cdot n$ flops, and method 2 requires $2000 + 200 \cdot n$ flops.

- (ii) Is it always better to use one of the two methods? Or does it depend on n ? If so, when should which method be used? (3 pts)

Solution: Method 1 is preferable when n is small. Method 2 should be used when $400 \cdot n > 2000 + 200 \cdot n$, which simplifies to $n > 10$.

2 Linear and least squares problems (40 pts)

2.1 Multiple choice (5 pts)

Circle all items that are true.

- A. A QR factorization tends to give more accurate results than the LU factorization when used to solve least squares problems.
- B. Ill-conditioned problems become better-conditioned when they are solved with a higher amount of precision.
- C. A matrix with numerical determinant 0 does not have an inverse.
- D. A nonlinear problem typically cannot be solved exactly in a finite number of steps, while a linear problem can.
- E. The forward error of an ill-conditioned problem is larger than the backward error.

Solution: A, D, E

2.2 Underdetermined systems and the Singular Value Decomposition (10 pts)

- (i) An *underdetermined* linear system has too few equations to reduce the solution space to a single point, hence many solutions exist. Complete the code below so that it returns a single solution \mathbf{x} of a linear system $\mathbf{Ax} = \mathbf{b}$ along with a list of vectors $\mathbf{y}_1, \mathbf{y}_2, \dots$ that span¹ the solution space. The line `U, S, Vt = svd(A)` performs a *full* (non-economy sized) SVD with the following signature:

$$\begin{matrix} \mathbf{A} & \mathbf{U} & \mathbf{S} & \mathbf{V}^T \\ \left[m \times n \right] & = & \left[m \times m \right] \left[m \times n \right] & \left[n \times n \right] \end{matrix}$$

Specifically, the singular values are returned in matrix form, and \mathbf{Vt} is equal to the transposed matrix \mathbf{V}^T . To find \mathbf{y} , it may be helpful to think of the SVD as a sum of outer products. (8 pts)

```
def solve_underdetermined(A, b):
    m, n = A.shape
    x = np.zeros(n)
    y = []
    U, S, Vt = svd(A)

    return x, y
```

¹in the sense that any linear combination can be added to \mathbf{x} , and the result still solves the linear system.

Solution: The missing portion:

```

for i in range(n):
    if i < m:
        x += (U[:, i] @ b) / S[i, i] * Vt[i, :]
    else:
        y.append(Vt[i, :])

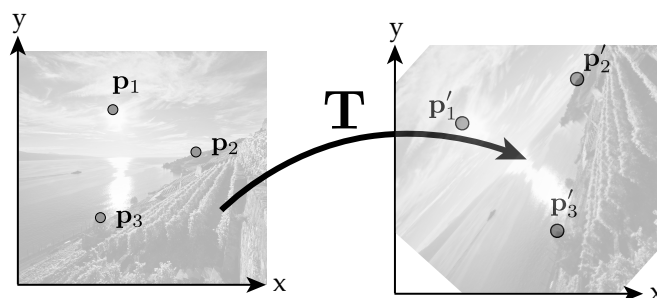
```

- (ii) Any kind of if-statement based on the rank or non-zero tests of $S[i, i]$ are of course also valid. What optimization problem does the point \hat{x} returned by your algorithm solve? Specify an expression involving "argmin". (2 pts)

Solution: $\operatorname{argmin}_x \|x\|^2$ subject to the constraint $Ax = b$.

2.3 Image alignment (17 pts)

After taking a brief break from editing your enormous photo collection, you realize in horror that a cat has walked over the keyboard of your computer while you were gone. All of the images are now scaled, rotated, and translated! Fortunately, you can still find the original of one of the images. To recover the exact transformation done by the cat, you decide to mark a number of matching points in both the original and transformed versions of the image:



The points are labeled

$$\begin{aligned}
 \mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n &\in \mathbb{R}^2 \text{ (original)} \\
 \mathbf{p}'_1, \mathbf{p}'_2, \dots, \mathbf{p}'_n &\in \mathbb{R}^2 \text{ (transformed)}
 \end{aligned}
 \tag{1}$$

We assume that \mathbf{p}'_i is related to \mathbf{p}_i via an affine transformation

$$\mathbf{p}'_i = T(\mathbf{p}_i) = \mathbf{T}_{linear} \cdot \mathbf{p}_i + \mathbf{t},
 \tag{2}$$

where $\mathbf{T}_{linear} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ is a linear transformation and $\mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$ denotes a translation. Your goal is to use the marked points to compute the transformation T .

- (i) Assuming that the points are well-chosen (e.g. not all in the same location), what is the smallest value of n that is necessary to recover T ? Is the associated problem exactly determined, overdetermined, or underdetermined? (1 pt)

Solution: At least three point pairs are needed. This results in an exactly determined problem with 6 unknowns $(a_{00}, a_{01}, a_{10}, a_{11}, t_x, t_y)$ and 6 equations—two for each point.

- (ii) A problematic aspect of affine transformations is that they aren't linear due to the additional translation term. Fortunately, affine transformations can be made linear by switching to *homogeneous coordinates*. Recall that this involves extending all points with an extra dimension that is set to the value one—for instance, $\mathbf{p}_i = (x_i, y_i)$ turns into $(x_i, y_i, 1) \in \mathbb{R}^3$.

Use homogeneous coordinates to re-write Equation 2 as a linear function, i.e. a single matrix-vector product $\mathbf{p}'_i = \mathbf{T} \mathbf{p}_i$. (2 pts)

Solution:

$$\begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

- (iii) We are now given n point pairs $(\mathbf{p}_1, \mathbf{p}'_1), \dots, (\mathbf{p}_n, \mathbf{p}'_n)$, where n is as determined in the first sub-problem and $\mathbf{p}_i = (x_i, y_i, 1)$ and $\mathbf{p}'_i = (x'_i, y'_i, 1)$. Write down a linear system, whose solution is the desired transformation T . (8 pts)

Solution:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{pmatrix} \begin{pmatrix} a_{11} \\ a_{12} \\ t_x \\ a_{21} \\ a_{22} \\ t_y \end{pmatrix} = \begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \\ y'_1 \\ y'_2 \\ y'_3 \end{pmatrix} \text{ (or any reordering of these equations)}$$

- (iv) To be on the safe side, you decide to mark a very large number of point pairs. Outline the necessary steps to solve the resulting least squares problem $\mathbf{Ax} \approx \mathbf{b}$ using a *QR factorization*.(4 pts)

Solution: QR allows us to directly solve the non-square system

$$\mathbf{Ax} = \mathbf{QRx} = \mathbf{b}$$

\mathbf{Q} is orthogonal and \mathbf{R} is upper triangular, so we solve by back substitution of the modified system

$$\mathbf{Rx} = \mathbf{Q}^T \mathbf{b}$$

- (v) After applying the inverse T^{-1} of the computed transformation to all of your images, you realize that they are still shifted and rotated! Further investigation reveals that a single one of the points pairs $(\mathbf{p}_i, \mathbf{p}'_i)$ was mixed up, creating an outlier that seems to have caused a large amount of error in the computation.

Why are least squares problems so sensitive to outliers? (2 pts)

Solution: Least squares solutions minimize the squared error of all equations. An outlier causes a large amount of error, which dominates all other equations after the errors are squared. This increased influence distorts the solution globally.

2.4 Conditioning and regularization (8 pts)

- (i) What are the characteristics of *ill-conditioned* problems, and why are they undesirable? How can we detect if a linear system is ill-conditioned? (3 pts)

Solution: Ill-conditioned problems are very sensitive to noise in the data and thus are challenging to solve with numerical techniques without the problem "exploding".

For linear systems, we can measure how much the output value of a function can change for a small change in the input with the *condition number*. (Low condition numbers indicate that a problem is well-conditioned while large condition numbers indicate that a problem is ill-conditioned.)

- (ii) Outline the necessary steps to apply Tikhonov regularization when solving least squares systems using the QR decomposition. (3 pts)

Solution:

QR: When using the QR factorization, we modify the system as follows:

$$\begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

- (iii) What optimization problem does the resulting value solve? Specify an expression involving "argmin". (2 pts)

Solution: $\operatorname{argmin}_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2 + \lambda \|\mathbf{x}\|^2$

3 Neural networks and optimization (30 pts)

3.1 Neural network basics (10 pts, 2 each)

The following questions refer to a standard neural network using fully connected layers with ReLU activation functions (as in Homework 5).

- (i) Specify the number of parameters/weights of a single neuron with 9 inputs.

Solution: $9+1 = 10$

- (ii) Specify the number of parameters/weights of a neural network with 9 inputs, 1 output, and a hidden layer with 100 neurons.

Solution: $10*100+101=1101$

- (iii) What is the main difference between regression and classification?

Solution: Two potential answers: Regression learns a function that approximates the data points, classification separates data points into individual groups. Regression is well suited for estimating continuous properties, classification for distinguishing discrete properties.

- (iv) What is the difference between standard gradient descent and stochastic gradient descent, and why is the stochastic version used with neural networks?

Solution: Only a random subset of the data is used to estimate the gradient. It would be prohibitively slow to compute the full gradient in each iteration.

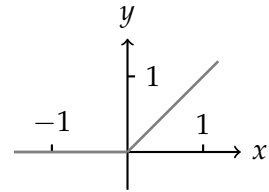
- (v) We discussed a sophisticated optimization methods in the course, including multivariate Newton's method and BFGS. Why is simple stochastic gradient descent used to train neural networks instead of these more advanced methods?

Solution: Two potential answers: These methods require (or approximate) the Hessian, which is too large since its size grows quadratically on number of parameters (which is $\gg 1000$ for most NNs). Training a neural network is an inherently noisy process, and noisy Hessian estimates will be too fragile.

3.2 Activation functions (8 pts, 4 each)

The structure of a standard neural network consists of neurons arranged in layers followed by activation functions—you will recall, e.g., the *rectified linear unit* (ReLU) activation function discussed in class:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



- (i) Suppose that the following *doubling linear unit* (DoLu) was used in a network that does classification:

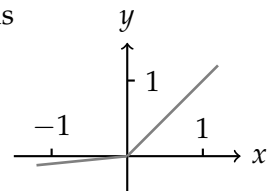
$$\text{DoLu}(x) = 2x$$

Do you expect this to have an influence on the classification performance compared to a network using ReLU activations? In either case, explain why.

Solution: Yes, performance will suffer significantly. When using DoLu, the entire network will be a concatenation of linear functions, i.e. no more powerful than a matrix multiplication.

- (ii) The ReLU activation function can be quite fragile during training, hence *leaky ReLU* (LReLU) activations are sometimes preferred. They are defined as

$$\text{LReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.1x & \text{otherwise} \end{cases}$$

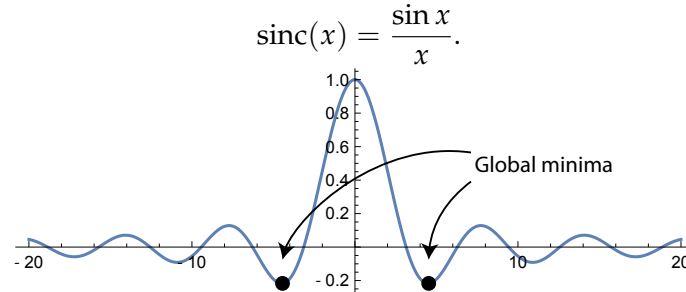


Explain the problem with ReLU that the leaky ReLU is meant to address.

Solution: Whenever the input of a ReLU unit is negative, it is effectively turned off. There is no direct way to recover from this situation, as the gradient will also be zero. A leaky unit will at least have a small gradient to ensure that the associated neuron participates in the optimization.

3.3 Nonlinear optimization (12 pts)

The *sinc* function commonly occurs in the area of signal processing and has the following simple definition (a plot on the interval $[-20, 20]$ is also shown).



Interestingly, there is no explicit formula for the two minima of this function—they can only be found numerically using optimization.

- (i) Derive all quantities that are needed to find minima of $\text{sinc}(x)$ using Newton's method (6 pts).

Solution:

$$\begin{aligned}\text{sinc}'(x) &= \frac{\cos x}{x} - \frac{\sin x}{x^2} \\ \text{sinc}''(x) &= -\frac{\sin x}{x} - \frac{\cos x}{x^2} - \frac{\cos x}{x^2} + \frac{2 \sin x}{x^3} \\ &= -\frac{\sin x}{x} - \frac{2 \cos x}{x^2} + \frac{2 \sin x}{x^3}\end{aligned}$$

- (ii) Complete the body of the following function so that it uses Newton's method to find a minimum given the starting point x . You can abbreviate functions determined in the previous answer (e.g. $\text{sinc}'(\dots)$) (3 pts).

```
def newton_sinc(x):
    for i in range(10):

        x -= sinc'(x) / sinc''(x)

    return x
```

- (iii) Mention two ways in which the above algorithm could fail (3pts).

Solution: The iteration could diverge when dividing by a very small second derivative. The algorithm could converge to any other local extremum (minimum or maximum).