First Name: **Signature:**

Last Name:

SCIPER ID: _____

EPFL

Final exam

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.

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. 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. There is an ε such that $(1+\varepsilon)+\varepsilon = 1$, but $1+(\varepsilon+\varepsilon) > 1$.

B. Nonsensical operations like subtracting infinity from infinity cause the application to terminate with an exception.

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. The identity `(x <= 0 || x > 0)` holds for every `x`.

1.2 Fused multiply-add (7 pts)

Recent processors include an instruction $\mathrm{fma}(a,b,c)$ 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).

(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 $\mathrm{fma}(a, b, c)$. 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)

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?

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

1.4 Matrix multiplication (8 pts)

Suppose that a computation must transform n 10-dimensional vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^{10}$ 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:

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

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

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

(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)

(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)

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.

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 $\mathbf{y}_1, \mathbf{y}_2, \ldots$ that span [1] the solution space. The line `U, S, Vt = svd(A)` performs a *full* (non-economy sized) SVD with the following signature:

$$\underset{\left[\, m \times n \,\right]}{\mathbf{A}} = \underset{\left[m \times m \right]}{\mathbf{U}} \underset{\left[\, m \times n \,\right]}{\mathbf{S}} \underset{\left[\, n \times n \,\right]}{\mathbf{V}^T}$$

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

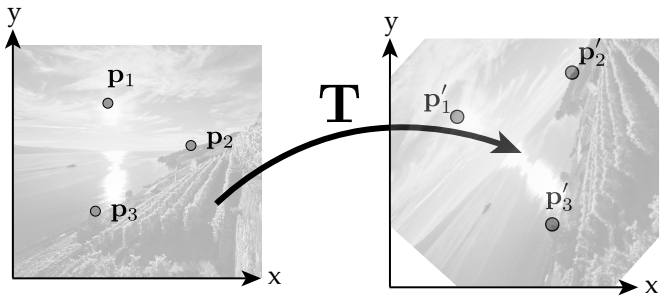
```
def solve_underdetermined(A, b):
    m, n = A.shape
    x = np.zeros(n)
    y = []
    U, S, Vt = svd(A)









    return x, y
```

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

(ii) What optimization problem does the point x returned by your algorithm solve? Specify an expression involving "argmin". (2 pts)

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

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

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)

(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)

(iii) We are now given n point pairs $(\mathbf{p}_1, \mathbf{p}'_1), \ldots (\mathbf{p}_n, \mathbf{p}'_n)$, 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)

(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 $\mathbf{A}\mathbf{x} \approx \mathbf{b}$ using a *QR factorization*.(4 pts)

(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)

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)

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

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

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.

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

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

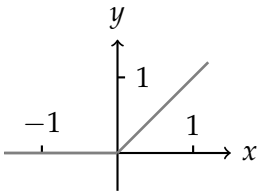
(iv) What is the difference between standard gradient descent and stochastic gradient descent, and why is the stochastic version used with neural networks?

(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?

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:

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



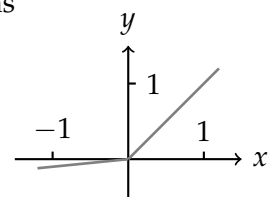
(i) Suppose that the following *doubling linear unit* (`DoLu`) was used in a network that does classification:

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

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

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

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

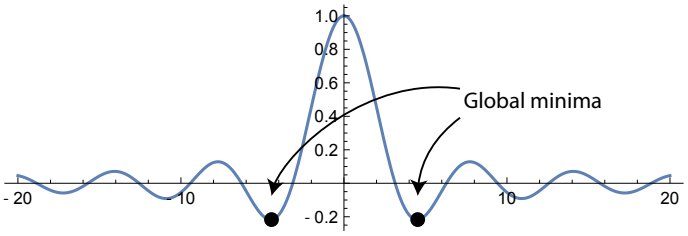


Explain the problem with ReLU that the leaky ReLU is meant to address.

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).

$$\text{sinc}(x) = \frac{\sin x}{x}.$$



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).

(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. $\text{sinc}'(...)$) (3 pts).

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




    return x
```

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