**First Name:** **Signature:**

**Last Name:**

**SCIPER ID:** _____

---

EPFL

# Practice exam

## CS328 - Numerical Methods in Visual Computing

### Procedures:

This is a timed examination. You will have a maximum of **2 hours** to complete this examination. Leave your **student ID card** on the table during the examination for verification. Write your **first and last name** as well as your **EPFL SCIPER number** on every sheet of paper. 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 on these sheets.

Multiple-choice items are each "worth" a fixed fraction of the question's score, while incorrect answers deduct a corresponding amount. The final score of multiple-choice question is clamped to a non-negative number.

### Regulations:

With the exception of one (1) page of personal notes written on a A4 page, any use of textbooks or other books/printed materials, formula sheets, calculators, and other similar aids are **NOT** permitted during the examination. The use of electronic devices such as laptops, cell phones or other PDAs, MP3 players, headphones, etc. is strictly **PROHIBITED** during the examination.

The student is deemed to have failed the course if he or she is found cheating or violating any of the above regulations.



---

| | | | |
|---|---|---|---|
| **Course Name:** | Numerical Methods in Visual Computing | **Date:** | 21.12.2016 |
| **Course Number:** | CS 328 | **Time:** | 14:15 - 16:00 |
| **Lecturer:** | Wenzel Jakob | | |

# 1 The IEEE754 standard

## 1.1 Breaking things

Write three brief Python snippets that cause numerical overflow, underflow, and create NaNs, respectively. Anything is allowed (except simply returning existing constants from the standard library).

```
def overflow():
```

```
def underflow():
```

```
def nan():
```

**Solution:**

```
def overflow():
    a = 1.0
    for i in range(1000):
        a *= 10.0
    return a

def underflow():
    a = 1.0
    for i in range(1000):
        a /= 10.0
    return a

def nan():
    return np.array(1.0) / np.array(0.0) * np.array(0.0)
```

### 1.2 Special values

In a few words, describe the purpose of the two special values `Inf` and `NaN` defined in the IEEE-754 standard.

**Solution:** `Inf` is by definition larger than any other representable floating point value, it marks computations containing a floating point overflow.
`NaN` signals an indeterminate/invalid computation (e.g. multiplication of `Inf*0`) and propagates through subsequent computations.

### 1.3 Denormalized numbers

In a few words, describe the purpose of denormalized numbers defined in the IEEE-754 standard.

**Solution:** Denormalized numbers postpone the effect of underflow by allowing computations involving very small values to continue at the finest possible floating point discretization instead of flusing them to zero.

### 1.4 Cancellation

Suppose that a program subtracts two numbers `x` and `y` having the same sign and very similar magnitudes, which (by chance) yields an *exact* answer $\texttt{z} = \texttt{x} - \texttt{y}$ that is not affected by any rounding errors. Does `z` suffer from cancellation? Justify your answer.

**Solution:** Cancellation involves the loss of significance when two very similar floating point numbers are subtracted from each other and the tiny residual with few significant digits moves to the leading position. Cancellation happens even when the result is exactly representable, hence: yes, `z` still suffers from cancellation.

### 1.5 Distributive law

Does floating point arithmetic satisfy the distributive law, i.e.

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

Justify your anwer.

**Solution:** IEEE765 guaranteees that the result of arithmetic operations such as $+$ and $\cdot$ matches that of an infinitely accurate computation that is rounded to the nearest floating point value. However, such guarantees don't extend to a more complex expression as the distributive law. For instance $b + c$ might overflow. If $a = 0$, the left hand side is `NaN`, and the right hand side is `Inf`.

## 2 Reverse engineering a nonlinear algorithm

You are reading some code from another programmer and you find numerical calculations without documentation and cryptic variable names. You eventually isolate the following iterative calculation that seems to be the most important part:

```
def compute_it(x):
    for i in range(10):
        x -= np.tan(x)
    return x
```

(i) What value does the iteration converge to? ? If there is more than one possible answer, any of them is fine

**Solution:** It converges to $n\pi$, where $n$ is any integer.

(ii) What algorithm is being used?

**Solution:** Newton's method.

(iii) What equation is being solved? Explain the code in terms of the standard equation for the algorithm.

**Solution:** $f(x) = \sin x$ x is being solved. The iteration formula for Newton's method is $x_{n+1} = x_n - \frac{f(x)}{f'(x)} = x_n - \frac{\sin x_n}{\cos x_n} = x_n - \tan x_n$.

## 3 Linear systems

We are solving a square linear system $\mathbf{A}\mathbf{x} = \tilde{\mathbf{b}}$, where $\tilde{\mathbf{b}} = \mathbf{b} + \mathbf{n}$ is contaminated by noise $\mathbf{n}$ with $\|\mathbf{n}\|_2 < \varepsilon$. We shall assume that the only errors in the computation are due to $\mathbf{n}$, and that rounding errors etc. are negligible. You can assume the singular value decomposition of $\mathbf{A}$ is available, with left and right singular vectors named $\mathbf{u}_i$ and $\mathbf{v}_i$ and singular values named $\sigma_i$.

(i) What is the largest possible absolute error in $\mathbf{x}$? In other words, give a bound on the difference between the solution with the true value of $\mathbf{b}$ and the solution computed from the noisy value of $\tilde{\mathbf{b}}$.

**Solution:** Call the true solution $\mathbf{x}$ and the perturbed solution $\mathbf{x} + \mathbf{m}$. The system we are solving is $\mathbf{A}(\mathbf{x} + \mathbf{m}) = \mathbf{b} + \mathbf{n}$. Since $\mathbf{A}\mathbf{x} = \mathbf{b}$, we can subtract $\mathbf{A}\mathbf{x}$ and $\mathbf{b}$ from the two sides of the previous equation, leaving $\mathbf{A}\mathbf{m} = \mathbf{n}$. So $\mathbf{m} = \mathbf{A}^{-1}\mathbf{n}$.
The operator norm of $\mathbf{A}^{-1}$ gives a bound on the error: $\|\mathbf{m}\| \leq \|\mathbf{A}^{-1}\|\|\mathbf{n}\|$. This norm is the largest singular value of $\mathbf{A}^{-1}$, which is the reciprocal of the norm of the smallest singular value of $\mathbf{A}$. This leads to $\|\mathbf{m}\| \leq \varepsilon/\sigma_n$.

(ii) What values of $\mathbf{b} \neq 0$ will lead to a large relative error in $\mathbf{x}$?

**Solution:** To make the relative error the largest, we need to make $\|\mathbf{x}\| = \|\mathbf{A}^{-1}\mathbf{b}\|$ as small as possible. We can do this with $\mathbf{b} = 0$, or assuming that $\mathbf{b} \neq 0$, the minimum $\mathbf{x}$ occurs when $\mathbf{b}$ is collinear with the first left singular vector $\mathbf{u}_1$.

(iii) What values of $\mathbf{n}$ will lead to a large relative error in $\mathbf{x}$?

**Solution:** The maximum error is achieved when $\mathbf{n}$ is collinear with the left singular vector $\mathbf{u}_n$ associated with the singular value $\sigma_n$.

### 3.1 Using factorizations

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ along with one of the factorizations shown below, explain how you would compute a solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, where $\mathbf{b} \in \mathbb{R}^n$. List all required steps and specify the associated big-O runtime complexity for each one (constant factors don't matter). Use the following format in your answer:

1. Invert $\mathbf{A}$ (cost: $O(...)$)
2. Multiply $\mathbf{A}^{-1}$ by $\mathbf{b}$ to obtain $\mathbf{x}$ (cost: $O(...)$)

You should assume that all involved steps are implemented efficiently, and that matrices with special structure (e.g. diagonal matrices) use an efficient encoding.

(i) $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ (Cholesky decomposition)

**Solution:**

1. Triangular solve using $\mathbf{L}$ (cost: $O(n^2)$)
2. Triangular solve using $\mathbf{L}^T$ (cost: $O(n^2)$)

(ii) $\mathbf{PA} = \mathbf{LU}$ (LU decomposition with partial pivoting)

**Solution:**

1. Apply permutation $\mathbf{P}$ to $\mathbf{b}$ (cost: $O(n)$)
2. Triangular solve using $\mathbf{L}$ (cost: $O(n^2)$)
3. Triangular solve using $\mathbf{U}$ (cost: $O(n^2)$)

(iii) $\mathbf{A} = \mathbf{QR}$ (QR decomposition)

**Solution:**

1. Multiply $\mathbf{b}$ by transpose of orthogonal matrix $\mathbf{Q}$ (cost: $O(n^2)$)
2. Triangular solve using $\mathbf{R}$ (cost: $O(n^2)$)

(iv) $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ (Singular value decomposition)

**Solution:**

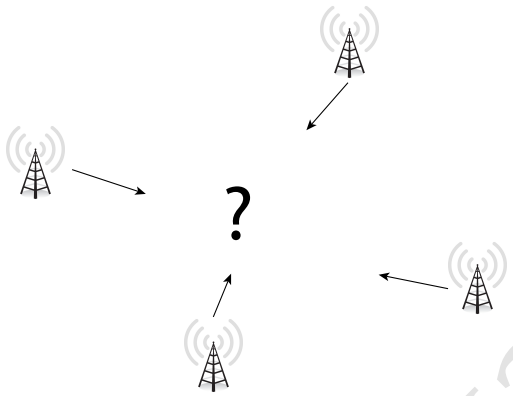
1. Multiply by transpose of left singular vector matrix $\mathbf{U}$ (cost: $O(n^2)$)
2. Multiply by inverse of diagonal matrix $\Sigma$ (cost: $O(n)$)
3. Multiply by right singular vector matrix $\mathbf{V}$ (cost: $O(n^2)$)

Are all approaches in the same complexity class, or is one of them asymptotically faster?

**Solution:** They are all in the same complexity class.

## 4 Radar localization

A signal source is briefly observed by $n$ radar dishes with *distinct* positions $(x_i, y_i)$ $(i = 1, \ldots, n)$ scattered on a 2D plane. Following the observation, the operator of each dish reports back a unit vector $(u_i, v_i)$ representing the most likely direction towards the signal.




We'll now focus on the problem of computing the source position $(s, t)$ given these observations.

(i) Suppose that $n = 3$. For what dish arrangement can we expect the associated numerical problem to be ill-conditioned? Does the problem still admit solutions in this case? Justify your answers.

**Solution:** If all radar stations are arranged on a line, they will not be able to localize sources on or very close to the line, hence the problem is ill-conditioned. However, sources at some distance can still be triangulated without problems.

(ii) Let's assume from now on that the radar dish locations were intelligently chosen to avoid the problems discussed in the previous question.

For what values of $n$ is the problem always underdetermined or overdetermined regardless of the source position? Why?

**Solution:** Each station leads to a constraint that requires the source to lie on a line. The problem is always underdetermined for $n = 1$, since a single line is not enough to determine a position. Since we are operating in two dimensions, the problem is always overdetermined for $n > 2$.

(iii) Let's assume that there are only two dishes. Construct a square linear system that can be used to solve for $(s, t)$.

**Solution:** The observation of each radar dish specifies that the signal source is located on a line, i.e.: $(s, t) = (x_i, y_i) + \alpha_i(u_i, v_i)$ (for some $\alpha_i \in \mathbb{R}$).
One solution approach is to stack the equations associated with both dishes into a $4 \times 2$ linear system and apply the normal equations to turn them into a square $2 \times 2$ system.
Alternatively, we can also express the parametric line equations in implicit form, i.e.

$$\begin{pmatrix} -u_i \\ v_i \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} -u_i \\ v_i \end{pmatrix} \cdot (s, t) \, .$$

which leads to the following $2 \times 2$ linear system:

$$\begin{pmatrix} -v_0 & u_0 \\ -v_1 & u_1 \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} u_0 y_0 - v_0 x_0 \\ u_1 y_1 - v_1 x_1 \end{pmatrix}$$



(iv) Now suppose that $n$ is very large. List three algorithms that can be used to solve this problem and discuss performance and accuracy tradeoffs that would need to be considered to choose a particular one.

**Solution:**

- Normal equations & LU decomposition: poor accuracy if linear system ill-conditioned.
- QR factorization: good accuracy if linear system is ill-conditioned; slower than LU.
- SVD: good accuracy if linear system is ill-conditioned; much slower than both LU and QR.

# 5 Numerical and analytic integration

## 5.1 Multiple-Choice

The following multiple-choice questions cover characteristic properties of analytic and numerical techniques covered in CS-328. Circle all items that are true.
You can assume that the functions in question are expressed using standard mathematical notation and only consist of elementary operations (addition, subtraction, square root, etc.)

A. The integral of a Riemann-integrable function can always be found using analytic techniques (i.e. calculus).

B. The derivative of a differentiable function can always be found using analytic techniques (i.e. calculus).

C. Integrating a quadratic polynomial using Simpson's rule will always yield an exact result.

D. A higher-order quadrature rule such as Simpson's rule is always more accurate than a lower-order quadrature rule such as the Trapezoid rule.

E. Monte Carlo integration is an effective technique to compute integrals of high-dimensional functions.

F. Monte Carlo integration requires that the function to be integrated is evaluated on a regular $n$-dimensional grid.

**Solution:** B, C, E

## 5.2 Orthogonal functions

Suppose that the following inner product is defined on the space of polynomials

$$\langle f, g \rangle \coloneqq \int_0^1 f(x)g(x)\,\mathrm{d}x.$$

In this abstract space, the two functions

$$f_1(x) \coloneqq x \qquad f_2(x) \coloneqq x + 1$$

span a two-dimensional "plane" (i.e. a subspace containing all linear combinations of $f_1$ and $f_2$).

(i) Apply the Gram-Schmidt algorithm using pencil & paper to compute an orthogonal basis $(f_1, \tilde{f}_2)$ of this plane.

**Solution:**

$$\int_0^1 x^2 \, \mathrm{d}x = \left[\frac{1}{3}x^3\right]_0^1 = \frac{1}{3}$$

$$\int_0^1 x(x+1) \, \mathrm{d}x = \left[\frac{1}{3}x^3 + \frac{1}{2}x^2\right]_0^1 = \frac{5}{6}.$$

Thus $f_1$ and

$$\tilde{f}_2(x) := f_2 - \frac{\langle f_1, f_2 \rangle}{\langle f_1, f_1 \rangle} f_1 = x + 1 - \frac{5}{2}x = 1 - \frac{3}{2}x$$

form an orthogonal basis.

(ii) Describe why orthonormal sets of functions are sometimes preferred to non-orthonormal ones (e.g. in the context of regression).

**Solution:** Non-orthonormal sets of functions can be (almost) linearly dependent, which causes the linear system used in regression to have an extremely large condition number. This in turn leads to inaccurate/unstable solutions.