Practice Exam
Issued: May 20, 2019, 10:15
Hand in: May 20, 2019, 12:00

Last Name:
First Name:
Student ID:

With your signature you confirm that you:

• Have read the exam directives
• You solved the exam without any unauthorized help
• You wrote your answers following the outlined directives

Signature:  

________________________________________________________________________

1
Exam directives. In order to pass the exam, the following requirements have to be met:

- Clear your desk (no cell phones, cameras, etc.): on your desk you should only have your Legi, your pen and your notes. We provide you with the necessary paper and exam sheets.

- Carefully read the first pages of the exam. Write your name and student ID where requested. Before handing in the exam, **SIGN ON THE FIRST PAGE**.

- Your notes (personal summary) should consist of no more than four A4 sheets (eight pages). The personal summary **must be handwritten**. You are not allowed to bring a copy of somebody else’s summary.

- Your answers should be handwritten in blue or black pen (no pencils), clearly readable and in English. Only one answer per question is accepted. Invalid answers should be clearly crossed out.

- To answer new questions (e.g. Question 1, not sub-questions!), always use a new page. On the top-right corner of every page write your complete name and Legi-ID. Unless otherwise noted in the question, you should always hand-in your answers on paper!

- You must hand in: the exam cover, any extra notes provided by us, the sheets with the exam questions and your solutions. The exam will not be accepted if any of these items are missing.

- If something is disturbing you during the exam or preventing you from peacefully solving the exam, please report it immediately to an assistant. Later complaints will not be accepted.
Question 1: Bayesian Inference (15 points)

Let $X$ be a random variable that depends on a parameter $\vartheta$. The conditional probability density function is given by

$$p(x \mid \vartheta) = \begin{cases} \frac{2x}{\vartheta^2}, & 0 < x < \vartheta \\ 0, & \text{otherwise} \end{cases}.$$  

Assume that you have observed one realization of the random variable $X = x_1$.

a) First, verify that $p(x \mid \vartheta)$ is a proper density function.

Obviously, $p(x \mid \vartheta) \geq 0$ for all $x \in \mathbb{R}$ and

$$\int_{\mathbb{R}} p(x \mid \vartheta) \, dx = \int_{0}^{\vartheta} \frac{2x}{\vartheta^2} \, dx = 1.$$  

b) Write the likelihood function $p(x_1 \mid \vartheta)$ as a function of $\vartheta$. You may want to use indicator functions: $\chi_{[a,b]}(x)$ is equal to 1 if $x \in [a, b]$ and 0 otherwise.

The likelihood function, can be written as,

$$p(x_1 \mid \vartheta) = \frac{2x_1}{\vartheta^2} \chi_{(0,\vartheta)}(x_1),$$

or, equivalently, as a function of $\vartheta$,

$$p(x_1 \mid \vartheta) = 2x_1 \vartheta^{-2} \chi_{(x_1,\infty)}(\vartheta) \chi_{(0,\infty)}(x_1), \quad (1)$$

where $\chi_{[a,b]}$ is the indicator function of the the set $[a, b]$.

c) Find the posterior distribution $p(\vartheta \mid x_1)$ for a uniform prior in $[0, 1]$.

Using Bayes' theorem,

$$p(\vartheta \mid x_1) = \frac{p(x_1 \mid \vartheta) p(\vartheta)}{p(x_1)}.$$  

(2)

The prior density is given by

$$p(\vartheta) = \chi_{[0,1]}(\vartheta).$$

First we note that if $x_1 \leq 0$ then the likelihood is zero and the posterior is zero.
Next, we consider the case $x_1 > 0$. The numerator in eq. (2), using eq. (1), is given by,

$$p(x_1 \mid \vartheta) p(\vartheta) = 2x_1 \vartheta^{-2} \chi_{(x_1, \infty)}(\vartheta) \chi_{(0, \infty)}(x_1) \chi_{[0, 1]}(\vartheta) = 2x_1 \vartheta^{-2} \chi_{(x_1, 1)}(\vartheta) \chi_{(0, 1)}(x_1). \quad (3)$$

The denominator $p(x_1)$ is given by

$$p(x_1) = \int_{\mathbb{R}} p(x_1 \mid \vartheta) p(\vartheta) \, d\vartheta = \int_{\mathbb{R}} 2x_1 \vartheta^{-2} \chi_{(x_1, 1)}(\vartheta) \, d\vartheta = 2x_1 \int_{x_1}^{1} \vartheta^{-2} \, d\vartheta = -2x_1 (\vartheta^{-1})\bigg|_{x_1}^{1} = 2(1 - x_1). \quad (4)$$

Finally, the posterior distribution is given by

$$p(\vartheta \mid x_1) = \begin{cases} \vartheta^{-2} \frac{x_1}{1 - x_1} \chi_{(x_1, 1)}(\vartheta), & 0 < x_1 < 1 \\ 0, & \text{otherwise} \end{cases}. \quad (5)$$

d) Estimate the uncertainty in the random variable $X$ after observing $x_1$ by calculating the conditional distribution of $X$ on $x_1$.

The posterior density of $X$ conditioned on $x_1$ is given by,

$$p(x \mid x_1) = \int_{\mathbb{R}} p(x \mid \vartheta) p(\vartheta \mid x_1) \, d\vartheta = \int_{\mathbb{R}} 2x \vartheta^{-2} \chi_{(x_1, \infty)}(\vartheta) \chi_{(0, \infty)}(x) \frac{x_1}{1 - x_1} \vartheta^{-2} \chi_{(x_1, 1)}(\vartheta) \chi_{(0, 1)}(x_1) \, d\vartheta = x \frac{2x_1}{1 - x_1} \chi_{(0, \infty)}(x) \chi_{(0, 1)}(x_1) \int_{x_1 \vee x}^{1} \vartheta^{-2} \, d\vartheta = x \frac{2x_1}{1 - x_1} \chi_{(0, 1)}(x) \chi_{(0, 1)}(x_1) \frac{1}{3} \vartheta^{-3}\bigg|_{x_1 \vee x}^{1} = x \frac{2x_1}{3(1 - x_1)} \left((x_1 \vee x)^{-3} - 1\right) \chi_{(0, 1)}(x) \chi_{(0, 1)}(x_1). \quad (6)$$
Question 2: Applied Optimization & UQ (20 points)

Rumours spread that you are amongst the most notorious programmers and statisticians in the region of Appenzell. The hedge fund *Appenance LTD* approaches you and wants you to have a look at their price prediction model $\mathcal{M}_1$ for cattle trading. They say that their strategy is presumably predicting wrong confidence intervals and hence is heavily underperforming. This particular cattle breed is traded over the counter only during weekends at the local markets in *Herisau* and for that reason trades data $d = \{d_i\}_{i=1}^N$ is very limited and difficult to collect.

Without blinking an eye, you accept their offer and you search for the parameters $\hat{\theta} = (\hat{x}_0, \hat{\sigma}^2)$ that maximize the likelihood of the model, i.e.,

$$\hat{\theta} = \arg \max_{\theta \in \Theta} L(\theta; d).$$

For obvious reasons the model can not be disclosed here, but we can tell that $x_0$ is a computational parameter and $\sigma^2$ describes the variance of the model error.

You use Korali’s optimizer (CMA-ES) to find the parameters that maximize the loglikelihood. After 4 consecutive runs you observe the following outputs on the screen:

**Listing 1: Run 1**

1. [Korali] CMA-ES Finished
2. [Korali] Optimum LogLikelihood (Maximize) found: $-8.49194e-01$
3. [Korali] Optimum LogLikelihood (Maximize) found at:
   4. $x_0 = +1.503e+00$
   5. $\sigma^2 = +3.501e+00$
6. [Korali] Stopping Criterium: Function value differences < $(1.00e^{-12})$

**Listing 2: Run 2**

1. [Korali] CMA-ES Finished
2. [Korali] Optimum LogLikelihood (Maximize) found: $-8.49193e-01$
3. [Korali] Optimum LogLikelihood (Maximize) found at:
   4. $x_0 = +3.004e+00$
   5. $\sigma^2 = +0.998e+00$
6. [Korali] Stopping Criterium: Function value differences < $(1.00e^{-12})$

**Listing 3: Run 3**

1. [Korali] CMA-ES Finished
2. [Korali] Optimum LogLikelihood (Maximize) found: $-8.49192e-01$
3. [Korali] Optimum LogLikelihood (Maximize) found at:
   4. $x_0 = +3.002e+00$
   5. $\sigma^2 = +0.999e+00$
6. [Korali] Stopping Criterium: Function value differences < $(1.00e^{-12})$
a) You observe that CMA-ES converges to different optima although the parameter of CMA-ES are kept the same in each run (except for the seed). Answer the following questions:

- Provide an explanation of the fact CMA-ES returns different results given the same objective function and configuration.
- What can you conclude about the problem regarding these results?
- Why does this pose a problem in predicting the price of cattle?
- Stochastic optimization is non-deterministic and hence may converge to different results, especially in multi-modal cases.
- The likelihood, respectively posterior distribution of the parameter (under the assumption of uniform priors) may be multimodal.
- Since we find two optima with different estimates for the model error the MLE choice has a direct impact on the estimates of the confidence intervals. If the full posterior distribution is taken into account, we would find confidence intervals that better predict the uncertainty of the model.

b) Assume that we know that the likelihood function is a multivariate gaussian mixture with local covariance $\Sigma$, i.e

$$L(\theta; d) = \sum_{i=1}^{2} \omega_i \mathcal{N}(\theta|\mu_i, \Sigma),$$

with $\omega_1 + \omega_2 = 1$ and

$$\Sigma = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}.$$

In order to further analyze the model at hand, you want to sample the posterior distribution of the parameters $\theta = (x_0, \sigma^2)$ given the data $d$,

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{p(d)}.$$
Since we know nothing about the parameters we choose a non informative prior distribution $p(\theta)$. What output do you expect from your sampling algorithm? Please provide 3 qualitative sketches and answer the questions:

- Define a prior distribution $p(\theta)$ for $x_0$ and $\sigma^2$.
- Specify the weights $\omega_i$ considering the output of CMA-ES from the previous subquestion.
- Sketch the sampling, i.e. draw a possible outcome of the samples in a scatter plot $x_0$ vs. $\sigma^2$. Include some contour lines of $p(\theta|d)$ if it supports the interpretability of your figure. Hint: the eigenvalue decomposition of $\Sigma$ is given by

\[
\Sigma = \frac{1}{\sqrt{2}} \begin{bmatrix}
-1 & 1 \\
1 & 1
\end{bmatrix} \cdot \begin{bmatrix}
0.5 & 0 \\
0 & 1.5
\end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix}
-1 & 1 \\
1 & 1
\end{bmatrix}^T.
\]

- Sketch the marginal distribution $p(\sigma^2|d) = \int_{-\infty}^{+\infty} p(\theta|d) \, dx_0$.
- Sketch the marginal distribution $p(x_0|d) = \int_{-\infty}^{+\infty} p(\theta|d) \, d\sigma^2$. 
Note: 3 sketches are required, 1 graph is spare. Don’t forget to label the axes, and add ticks to the axes where possible (vertically and horizontally) in order to localize your graphs. If you make assumptions, please document it. Please make sure that your plots are consistent throughout.

- Weights: $\omega_i = 0.5$. The likelihood is the same at both modes (assuming uniform prior).
- Bounds: here, non informative prior means uniform prior with bounds $(-\infty, +\infty)$ for $x_0$ and $(0, +\infty)$ for $\sigma^2$.
- Axes: modes in all three plots should be clearly around $(x_0 = 3.0, \sigma^2 = 1.0)$ and $(x_0 = 1.5, \sigma^2 = 3.5)$
- Labels: clearly describe what you plotted on the $x$ and $y$ axis in all three plots.
- Marginal Distributions: Two modes with roughly the same width. Location of the modes are at $(1.5, 3.0)$ for $p(x_0)$ and at $(1.0, 3.5)$ for $p(\sigma^2)$. Marginal distribution must match the drawn samples, respectively contour lines, from the scatter plot. Verify in both sketches if the valley between the modes should reach zero or not, corrected individually per student. The marginal distribution must be greater 0, but the height is not given.
Students should not blindly overtake values from CMA-ES results (e.g. log-likelihood).

- **Sampling:** Two accumulation points of samples around the modes, samples should reflect $\Sigma$, i.e. the samples are not circularly distributed around the modes but ellipsis-shaped. The inclination angle of the ellipses must be visible. The main axes correspond to the eigenvectors of $\Sigma$. The longer corresponds to the eigenvector $(1, 1)^T$. The ratio of the length of the axes, respectively the spread of the samples, is not expected to be correct (correct ratio is $\sqrt{1.5}/\sqrt{0.5} = 1.7$).

- **Contour Line:** (optional) contour lines drawn in sampling plot, contour lines should be consistent with the sampling. If they assign values to the contour lines (e.g. log-likelihood) taken from CMA-ES results it counts as a mistake (without further reasoning).
c) In the meantime the research department of *Appenance LTD* came up with a more sophisticated pricing model $M_2$, taking into account the PH-value of the grass the cows eat and the amount of days they graze in rain. You discuss this model with your friend Reto Küehli who is a cow specialist and he assures that these parameter have no influence on the development of the cows. Nevertheless the researchers find that the maximal likelihood for the new model $M_2$ is larger than the maximal likelihood for $M_1$ (on data set $d$).

Answer the following questions in 3-5 sentences each:

- How can you explain that model $M_2$ achieves a “better” likelihood although it includes irrelevant factors.
- What other measures do you know from this course that allow you to select between $M_1$ and $M_2$? Also argue why comparing the likelihoods of $M_1$ and $M_2$ is not best practice and why you would choose a different measure.
- A model with more parameters can have more inflection points, so of course comes closer to the data points and hence increase the likelihood.
• Model selection based on evidence $p(d)$. Evidence balances the decrease in likelihood with the increase in the number of parameters. Other possibilities are AIC, BIC.
Question 3: Parallel Tasking Theory (20 points)

You work at a newly instituted supercomputing center and every morning you receive a queue of 12 jobs that need to be executed. Each job can be executed independently and cannot be parallelized. The computational cost of each job is given in the following in terms of a reference computational load \( C \), but you do not know this load a-priori to design an optimal parallel tasking strategy. Communication costs are ignored in this exercise.

<table>
<thead>
<tr>
<th>Job order</th>
<th>Comp. load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
<td>12 1 1 12 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

We assume for simplicity that this load can be directly translated to a fixed reference computational time, e.g. that a job of load \( C = 7 \) can be executed by a single rank in \( T = 7 \) time units. Time measurements in the following need to be provided in terms of these time units. You are provided with a small compute node with 4 cores, each capable of running a single job at a time, and you are asked to propose algorithms taking advantage of various parallel tasking strategies.

a) In the classical divide-and-conquer method the tasks are divided to the available processors. You cannot affect the ordering of the jobs. This leads to the following partition:

<table>
<thead>
<tr>
<th>Job order</th>
<th>Comp. load</th>
<th>Exec. by rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
<td>12 1 1 12 1 1 1 1 1 1 1 1</td>
<td>1 1 1 2 2 2 3 3 3 4 4 4</td>
</tr>
</tbody>
</table>

Compute:

1. the total runtime \( T_{total} \)
2. the average rank run-time \( T_{avg} \)
3. the load imbalance ratio \( I \)
4. the average rank idle time \( W_{avg} \).

The total run-time is given by the slowest rank in this case. \( T_{total} = 14 \). The rank times of each rank are \( T_1 = T_2 = 14 \) and \( T_3 = T_4 = 3 \). The average rank time is \( T_{avg} = (14 + 14 + 3 + 3)/4 = 8.5 \). The load imbalance ratio is \( I = (T_{total} - T_{avg})/T_{total} = (14 - 8.5)/14 \approx 0.4 = 40\% \). The idle times of
the ranks are \( W_1 = W_2 = 0, W_3 = W_4 = 11 \). This leads to an average idle time of \( W_{avg} = 22/4 = 5.5 \) time units. As a consequence the answers are:

1. \( T_{total} = 14 \)
2. \( T_{avg} = 8.5 \)
3. \( I = 40\% \)
4. \( W_{avg} = 5.5 \)

b) Now you are asked to apply the producer-consumer strategy. Note that you cannot affect the ordering of the jobs assigned to the ranks. Which rank will execute each job? Please fill in the following array and answer the questions that follow:

\[
\begin{array}{cccccccccccc}
\text{Job order:} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\text{Comp. load:} & 12 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{Exec. by rank:} & & & & & & & & & & & & \\
\end{array}
\]

1. What is the total runtime \( T_{total} \) in this case?
2. What is the average rank run-time \( T_{avg} \)?
3. What is the load imbalance ratio \( I \)?
4. What is the average rank idle time \( W_{avg} \)?
5. What is the disadvantage of the producer-consumer strategy in general? Is the extra effort to implement this strategy worth it in this case?

The jobs are assigned to the next available rank by the master rank. We assume that the first rank is the master (producer) and is responsible for the communication of the jobs. This leads to the following distribution:

\[
\begin{array}{cccccccccccc}
\text{Job order:} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\text{Comp. load:} & 12 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{Exec. by rank:} & 2 & 3 & 4 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\
\end{array}
\]

The first job is assigned to rank 2. This computationally heavy job will need 12 time units to run. As a consequence, rank 2 is busy for the rest of the program execution time. The master then assigns jobs 2 and 3 to ranks 3 and 4 respectively. In job 4, the master waits for the next available rank. This will logically be rank 3 (a solution with rank 4 is also acceptable). The master assigns job 4 to rank 3. As job 4 is a computationally heavy job, rank 3 is
also busy till the end of the execution. The master iteratively assigns the rest of the computationally light jobs to rank 4, which is the only available rank. The execution times are $T_2 = 12$, $T_3 = 12 + 1 = 13$, and $T_4 = 9$. The total execution time, is determined by the slowest rank, which is rank 3, $T_{\text{total}} = 13$. The average execution time is $T_{\text{avg}} = (12 + 13 + 9)/3 = 11.33$.

The load imbalance is $I = (13 - 11.33)/13 = 0.128 \approx 0.13 = 13\%$. The idle times of each rank are $W_2 = 1, W_3 = 0, W_4 = 4$. The average idle time is $W_{\text{avg}} = (4 + 1)/3 = 1.66$.

1. $T_{\text{total}} = 13$
2. $T_{\text{avg}} = 11.33$
3. $I \approx 13\%$
4. $W_{\text{avg}} = 1.66$
5. The disadvantage of the producer-consumer strategy is that one whole rank is devoted to the communication of the jobs to the next available rank. In this case, the load imbalance ratio and the total execution time is improved. As a consequence, we can argue that the extra effort to implement this strategy is worth it.

**Important:** Note that the producer task cannot be devoted to processing tasks itself because it needs to be ready to provide new work to consumers as soon as they become ready.

We can show how destructive the effect of such approach (i.e., having the producer task be also a consumer) would be by imagining, for example, that the producer task is unlucky enough to start processing a really big job, while all the consumers work on smaller tasks. Until the producer (now also consumer) finishes with its big job, it will not be able to provide new work to the (actual) consumer tasks. Therefore, all but one of the available cores will have to wait idling while the producer finishes, wasting much more computation time than it would if the core was just dedicated to producing. This problem is much more pronounced the more cores we use so, even if communication costs are null, we should always leave the producer task free to deliver jobs immediately on demand.

c) The problem with the jobs is that the computational load is not balanced and we do not know the load a-priori to design an optimal tasking strategy. A different ordering of the same jobs might cause the worst case scenario (in terms of run-time) for a divide-and-conquer tasking strategy.
1. What is an \textbf{example} of an ordering that causes this worst case scenario? To which rank is each job assigned in your example?

2. Assuming that the jobs are assigned a random uniform ordering by an external client, what is the probability of a worst case scenario occurring?

3. How much faster is the producer-consumer strategy in this case? Report the speed-up compared to the divide-and-conquer method.

A worst case scenario occurs when by chance, both computationally heavy jobs are assigned to the same rank. An example of such an ordering is

\begin{center}
\begin{tabular}{cccccccccccc}
Job order: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12
\hline
Comp. load: & 1 & 1 & 1 & 1 & 12 & 12 & 1 & 1 & 1 & 1 & 1 & 1
\hline
Exec. by rank: & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 4
\end{tabular}
\end{center}

where rank 2 has to compute both computationally heavy jobs.

The probability of a worst case scenario, is the probability that the two intensive jobs be assigned to the same rank. The total number of different permutations of the $k=2$ computationally intensive jobs in the position of the $n=12$ total jobs is given by the binomial formula:

\begin{equation}
\begin{aligned}
\binom{n}{k} &= \frac{n!}{k! \cdot (n-k)!} = \frac{12!}{10! \cdot 2!} = \frac{11 \cdot 12}{2} = 11 \cdot 6 = 66.
\end{aligned}
\end{equation}

For a fixed rank out of the total 4, there are three cases of the job permutation in the rank that indicate a worst case scenario:

\begin{center}
\begin{tabular}{ccc}
Case 1 & 1 & 12 & 12 \\
Case 2 & 12 & 1 & 12 \\
Case 3 & 12 & 12 & 1 \\
\end{tabular}
\end{center}

This means that in total we have $4 \times 3$ total permutations that lead to worst-case scenarios. The probability of a worst-case scenario is therefore

\begin{equation}
P = \frac{12}{66} = \frac{2}{11} = 0.18
\end{equation}

The total run-time of worst-case scenario in the divide-and-conquer method is $T_{\text{total}}^{D&C} = 12 + 12 + 1 = 25$ time units. The total run-time of the producer-consumer strategy in the same scenario is $T_{\text{total}}^{P&C} = 12 + 1 + 1 + 1 = 15$ time units.
units. As a consequence the speed-up is

\[ \text{Speedup} = \frac{T_{P&C}}{T_{D&C}} = \frac{25}{15} = 1.66 \]  \hspace{1cm} (9)

**Grading Scheme**

- **a)** 4 points. One point for each correct answer.
- **b)** 6 points. One for each correct answer, in 5. one point for disadvantage, one point for answering if it is worth it (yes).
- **c)** 10 points. 1 point for correct example, 1 point for correct job assignment. 6 points for correct probability (2 pts for estimating the 12 cases, 2 pts for estimating the 66 cases, and 2 pts for the correct probability). 2 pts for correct speed-up.
Question 4: Parallelizing the Quadratic Form (20 points)

In this exercise, you are asked to parallelize a quadratic form with UPC++. The quadratic form is given by

\[ Q = v^T \cdot A \cdot w = \sum_{ij} v_i A_{ij} w_j \]  

(10)

where the matrix \( A \in \mathbb{R}^{n \times n} \) and \( v, w \in \mathbb{R}^n \). The entries of \( A, w, \) and \( v \) are given by:

\[ A_{ij} = \frac{i + 2j}{n}, \quad v_i = 1 + \frac{2}{i + 0.5}, \quad w_i = 1 - \frac{i}{3} \]  

(11)

In the following you are provided with the serial program.

```c
#include <stdio.h>
#include <vector>
#include "matrix.hpp"

int main(int argc, char** argv) {

    int n = 1024;

    hpcse::matrix<double, hpcse::row_major> A(n, n);
    std::vector<double> v(n), w(n);

    for (int i=0; i<n; ++i)
        for (int j=0; j<n; ++j)
            A(i,j) = (i + 2.*j) / n / n;

    for (int i=0; i<n; ++i)
        v[i] = 1. + 2. / (i + 0.5);

    for (int i=0; i<n; ++i)
        w[i] = 1. - i / 3. / n;

    double result = 0.;
    for (int i=0; i<n; ++i)
        for (int j=0; j<n; ++j)
            result += v[i] * A(i,j) * w[j];

    if(rankId==0)
        printf("Rank 0  -  Total result \%lf.\n", result);

    upcxx::finalize();
}
```
a) Fill in the 20 missing **gaps** after the TODO: comments in the following parallel implementation.

```c
#include <stdio.h>
#include <vector>
#include "matrix.hpp"
#include <upcxx/upcxx.hpp>

int main(int argc, char ** argv) {
    upcxx::init();
    int rankId = upcxx::rank_me(), rankCount = upcxx::rank_n();
    int n = 1024, chunk = n / rankCount;

    // TODO: Initialize a global pointer with name globalResults
    ___________________ globalResults;
    if(rankId==0) globalResults = ______________<double>(
        ___________________);

    // TODO: Broadcast the adress of the global pointer to all ranks
    ______________________(_____________________, 1, 0).wait();

    hpcse::matrix<double, hpcse::row_major> local_A(chunk,n);

    // TODO: Define local_v and w as vectors of doubles of the correct size
    std::vector<double> local_v(____________), w(____________);

    for (int i=0; i<chunk; ++i)
        for (int j=0; j<n; ++j)
            local_A(i,j) = ((rankId*chunk + i) + 2.*j) / n / n;

    for (int i=0; i<chunk; ++i)
        // TODO: Fill in the entries of the local v vector
        local_v[i]=1.+2./( (__________+_________+______) );

    for (int i=0; i<n; ++i) w[i]=1.-i/3./n;

    double local_result = 0.;
    for (int i=0; i<chunk; ++i)
        for (int j=0; j<n; ++j)
            // TODO: Add the result to the local result
            __________ += __________ * local_A(i,j) * __________;

    // TODO: Update the globalResults pointer with the local result
    __________(__________,globalResults + __________, 1).wait();
    upcxx::barrier();

    if (rankId == 0){
        double global_result = 0.;
        // TODO: Enable access of the global results pointer locally
        double* __________ = globalResults.__________;
        for (int i = 0; i < rankCount; i++){
```
// TODO: Add each rank result to the global result

result += result;
}
printf("Rank 0 - Total result %lf.\n", global_result);
upcxx::finalize();
```c
#include <stdio.h>
#include <vector>
#include "matrix.hpp"
#include <upcxx/upcxx.hpp>

int main(int argc, char** argv) {
    upcxx::init();
    int rankId = upcxx::rank_me(), rankCount = upcxx::rank_n();
    int n = 1024, chunk = n / rankCount;

    // TODO: Initialize a global pointer with name globalResults
    upcxx::global_ptr<double> globalResults;
    if (rankId == 0) globalResults = upcxx::new_array<double>(rankCount);
    // TODO: Broadcast the address of the global pointer to all ranks
    upcxx::broadcast(&globalResults, 1, 0).wait();

    hpcse::matrix<double, hpcse::row_major> local_A(chunk,n);
    // TODO: Define local_v and w as vectors of doubles of the correct size
    std::vector<double> local_v(chunk), w(n);

    for (int i=0; i<chunk; ++i)
        for (int j=0; j<n; ++j)
            local_A(i,j) = ((rankId * chunk + i) + 2.*j) / n / n;

    for (int i=0; i<chunk; ++i)
        // TODO: Fill in the entries of local v vector
        local_v[i]=1.+2./((rankId*chunk+i)+0.5);

    for (int i=0; i<n; ++i) w[i]=1.-i/3./n;

    double local_result = 0.;
    for (int i=0; i<chunk; ++i)
        for (int j=0; j<n; ++j)
            // TODO: Add the result to the local result
            local_result += local_v[i] * local_A(i,j) * w[j];

    // TODO: Update the globalResults pointer with the local result
    upcxx::rput(&local_result, globalResults + rankId, 1).wait();
    upcxx::barrier();

    if (rankId == 0){
        double global_result = 0.;
        // TODO: Downcast the global results pointer
        double* local_result = globalResults.local();
        for (int i = 0; i < rankCount; i++){
            // TODO: Add each rank result to the global result
            global_result+= local_result[i]; // Adding to the global result
        }
        printf("Rank 0 - Total result %lf\n", global_result);
    }
    upcxx::finalize();
}
```
b) What is the parallel tasking strategy used to parallelize the quadratic form?

The paralleling tasking strategy used is the divide-and-conquer strategy.

c) Comment on the load balance of each rank. Do you think the strategy employed is reasonable, or another strategy might be much more efficient for this task?

The task load that each rank has to compute is weighted, as each rank has to compute the same number of column sums, and the computation load of each column sum is approximately the same. For this reason, a divide-and-conquer strategy is a reasonable selection.

d) The parallel implementation contains a barrier in line 43. Do you think this barrier is needed? Why?

Yes, the barrier is needed to make sure all ranks have finished the computation of the tasks and placement of the results on the global pointer, before the master rank computes the quadratic form. If this barrier is omitted, racing issues appear, as it might be the case that the master rank computes its task faster and computes the quadratic form before other ranks have written into the global results pointer. This implies a racing condition.

Grading Scheme

- a) 15 points for exercise, 0.75 for every gap.
- b) 1 point.
- c) 2 point. One for load balanced, one for reasonable.
- d) 2 point. One for answer, one for explanation.
Question 5: GEMV on GPUs (20 points)

The code below computes a matrix-vector product (GEMV) using an \( N \times N \) matrix

\[
a = M b, \quad \text{where} \quad a_i = \sum_{j=0}^{N-1} M_{i,j} b_j.
\]  \hspace{1cm} (12)

```cpp
#include <vector>
#define VECSIZE 512

void random_init(std::vector<float> & v, std::vector<float> & m)
{
    // Some random initialization - this could be anything
    for(std::size_t i = 0; i < v.size(); ++i)
        v[i] = i;
    for(std::size_t i = 0; i < m.size(); ++i)
        m[i] = i;
}

void gemv_cpu(float * a, float * m, float * b, unsigned int N)
{
    for(unsigned int i = 0; i < N; ++i)
        for(unsigned int j = 0; j < N; ++j)
            a[i] += m[N*j+i] * b[j];
}

int main()
{
    unsigned int const N = VECSIZE;
    std::vector<float> a(N);
    std::vector<float> b(N);
    std::vector<float> matrix(N*N);
    random_init(b, matrix);

    // TODO replace gemv_cpu.
    // The code should perform the matrix-vector multiplication
    // on the GPU and store the result in the CPU vector 'a'.
    //
    gemv_cpu(&a[0], &matrix[0], &b[0], N);

    return 0;
}
```

a) Modify the host code such that the matrix-vector product is computed on the GPU. The final result should be available on the CPU. Write and call a CUDA kernel (gemv_gpu) for square matrices of arbitrary dimension \( N \).
Notes:

- Show how the kernel is called in the main() function.
- Checking CUDA for errors is not mandatory.

```cpp
#include <vector>
#define VECSIZE 512
#define THREADS_PER_BLOCK 32

// CUDA kernel
__global__ void gemv_gpu(float * a, float * m, float * b, unsigned int N)
{
    unsigned int const i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i < N)
    {
        a[i] = 0.;
        for(unsigned int j=0; j < N; ++j)
            a[i] += m[N*j+i] * b[j];
    }
}

int main()
{
    unsigned int const N = VECSIZE;
    std::vector<float> a(N);
    std::vector<float> b(N);
    std::vector<float> matrix(N*N);

    random_init(b,matrix);

    // Host code (part a)
    unsigned int const num_threads = THREADS_PER_BLOCK;
    unsigned int num_blocks = ceil(double(N)/double(num_threads));
    float *d_a, *d_b, *d_m;
    cudaMalloc(&d_a, a.size() * sizeof(float));
    cudaMalloc(&d_b, b.size() * sizeof(float));
    cudaMalloc(&d_m, matrix.size() * sizeof(float));
    cudaMemcpy(d_b, &b[0], b.size() * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_m, &matrix[0], matrix.size() * sizeof(float), cudaMemcpyHostToDevice);

    gemv_gpu<<<num_blocks, num_threads>>>(d_a, d_m, d_b, N); // (b), kernel call

    // Host code (part b)
    cudaMemcpy(&a[0], d_a, a.size() * sizeof(float), cudaMemcpyDeviceToHost);
}

cudaFree(d_m);
```
cudaFree(d_b);
cudaFree(d_a);

return 0;
}
Question 6: Advanced MPI: Communicators (30 points)

The $N \times N$ Lehmer matrix $A$ is defined as $A_{i,j} = \min(i,j)/\max(i,j)$, where $i,j = 1,2,\ldots,N$. In this task you will compute the $L_{\infty}$ norm

$$L_{\infty} = \max_i \left\{ \sum_j |A_{ij}| \right\}$$

of this matrix distributed across a two dimensional grid of MPI processes.

a) In the first part of this task you have to complete the following code snippet to initialize the matrix. You have 16 available MPI processes, and should divide the matrix in a $4 \times 4$ grid. You can assume that the size of the matrix $N$ is always divisible by 4. Modify the code between TODO_START and TODO_END in order to initialize the matrix using an MPI cartesian grid topology.

```c
int size, rank, size_x, size_y;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

// TODO_START
// assume that we have size = 16 ranks in total
// then size_y = size_x = 4
size_x = 4;
size_y = size / size_x;
assert(N % size_x == 0 && N % size_y == 0);

// local matrix size
const int Nx_loc = N / size_x;
const int Ny_loc = N / size_y;
const int chunk_size = Nx_loc * Ny_loc;

// compute coordinates (i,j) of rank
int coord_x = rank / size_x;
int coord_y = rank % size_x;
// TODO_END

double* matrix_loc = new double[chunk_size];

// initialize local matrix
for(int i = 0; i < Ny_loc; ++i)
    for(int j = 0; j < Nx_loc; ++j)
        {
            // compute global indices
            int i_glo = coord_y*Ny_loc + i;
            int j_glo = coord_x*Nx_loc + j;
            matrix_loc[i*Nx_loc + j]
```
Grading scheme:

4 pts for defining dims, size_x, size_y, full points given for using the original 4x4 split
4 pts for creating the cartesian grid correctly
4 pts for creating and defining the coords

b) A code snippet used for the computation of the infinity norm $L_\infty$ is provided below. Complete the code between the two TODO / END_TODO "brackets" in order to perform the reductions using row and column communicators.

```cpp
// compute $L_\infty = \max_{col} \left( \sum_{row} (a_{ij}) \right)$

double sums_loc[Ny_loc]; // local sum
for(int i = 0; i < Ny_loc; ++i) {
    double sum = 0.0;
    for(int j = 0; j < Nx_loc; ++j)
        sum += std::abs(matrix_loc[i*Nx_loc + j]);
    sums_loc[i] = sum;
}

double sums_row[Ny_loc]; // overall sums of rows

// TODO: create row communicator and perform sum reduction

// END_TODO

double max_strip = 0.0; // local max of every sums_row
for(int i = 0; i < Ny_loc; ++i)
    max_strip = std::max(max_strip, sums_row[i]);

double L_inf = 0.0; // overall max

// TODO: column communicator and max reduction with it
```
if (rank == 0)
    std::cout << "L_inf norm: " << L_inf << std::endl;

Solution codes:

// TODO: create row communicator and perform sum reduction
int which_row = rank % size_y;
MPI_Comm row_comm;
MPI_Comm_split(MPI_COMM_WORLD, which_row, rank, &row_comm);
for (int i = 0; i < Ny_loc; ++i)
    sums_row[i] = 0.0;
MPI_Reduce(sums_loc, sums_row, Ny_loc, MPI_DOUBLE, MPI_SUM, 0, row_comm);

// END_TODO

// TODO: column communicator and max reduction with it
int which_column = rank / size_y;
MPI_Comm column_comm;
MPI_Comm_split(MPI_COMM_WORLD, which_column, rank, &column_comm);
MPI_Reduce(&max_strip, &L_inf, 1, MPI_DOUBLE, MPI_MAX, 0, column_comm);

// END_TODO

Grading scheme:

18 pts:

9 pts for row communicator: 2 pts for identifying the row of rank, 4 pts for creating the communicator, 3 pts for the sum Reduction
9 pts for col communicator: 2 pts for identifying the col of rank, 4 pts for creating the communicator, 3 pts for the max Reduction
Full points were given to answers using MPI_Cart_split instead, if they used x as remain dimension for rows