STATISTICAL COMPUTATION & SIMULATION
HOMEWORK 1

Assigned January 17, 2026, due February 17, 2026

This homework pertains to materials covered in the first month of the semester. Your final report need to be typed, with your team member names listed on the document, and with properly labeled computer output for those problems that require computer simulations. To obtain full credit, I suggest you embed your (`R`, `python`, or whatver you are using) input code within the main text and organize them well so they are easy to read. You also need to attach your code on Canvas as a separate file and clearly indicate the problem it corresponds to. Please also describe your work and findings with sufficient details. Upload the file to the Homework 1 on Canvas or hand it in before class on Feb 17.

While AI tools (e.g., ChatGPT) might help you explore ideas, they are not a substitute for deriving results yourself. Submissions that rely on AI outputs or on solutions from previous years will receive a zero, and such cases will be reported to the Office of Academic Integrity.

Problem 1 (Matrix inversion; 15 points)**.** Consider matrices of the form $\Sigma = \sigma I + KK^T$ where K is an $N \times M$ matrix of i.i.d. $N(0,1)$ random variables and $\sigma = 0.2$. Fix $M = 10$. Compute the inverse of the matrix using two different algorithms:

(1) directly by using the `solve()` function in `R`;
(2) using the Sherman-Morrison-Woodbury identity discussed in class.

I have provided example code for simulating the matrix and timing the solve function here

```
M <- 10; sigma <- 0.2
N <- 100
K <- matrix(rnorm(N*M), nrow=N, ncol=M)
Sigma <- sigma*diag(rep(1,N))+K%*%t(K)
cpu_time <- system.time(solve(Sigma))
print(cpu_time)
```

Report the following:

(a) Plot the CPU time versus N for algorithm (1) and algorithm (2) (you will have to decide the grid and how large you can go with N);
(b) A plot of floating point operations (flops) versus N for algorithm (1) and algorithm (2);

(c) Briefly summarize what you observe based on a comparison of the computational costs for the two algorithms using flops and using CPU time.

Problem 2 (SVD; 50 points + 10 bonus points)**.** Risser et al. [2025] developed a spatio-temporal extreme-value model to estimate the *theoretical upper bound* of temperature at a given location from historical records at weather stations. The file `Temp_data.RData` contains a global dataset of temperature upper bounds evaluated at $n = 37{,}511$ weather stations. Let

- $\mathbf{y} \in \mathbb{R}^n$ be the vector `Temp_UB` (upper bounds in °C),
- $\mathbf{X} \in \mathbb{R}^{n \times p}$ be the matrix `Xmat` with $p = 3{,}535$ columns, each column $\mathbf{X}_{\cdot j}$ represents a *localized basis function* (a spatially local "bump" supported around a region),
- `stations` be a data frame that contains the longitude-latitude coordinates of the stations.

Our goal is to approximate the high-dimensional `Temp_UB` field via a basis expansion

$$\mathbf{y} \approx \mathbf{X}\boldsymbol{\beta},$$

where $\boldsymbol{\beta} \in \mathbb{R}^p$ are coefficients. The top right panel of Figure 1 shows an example reconstruction using $\mathbf{X}\boldsymbol{\beta}$. Since $p \ll n$, it is often convenient to work with the lower-dimensional coefficient vector $\boldsymbol{\beta}$ instead of the much higher dimensional input.

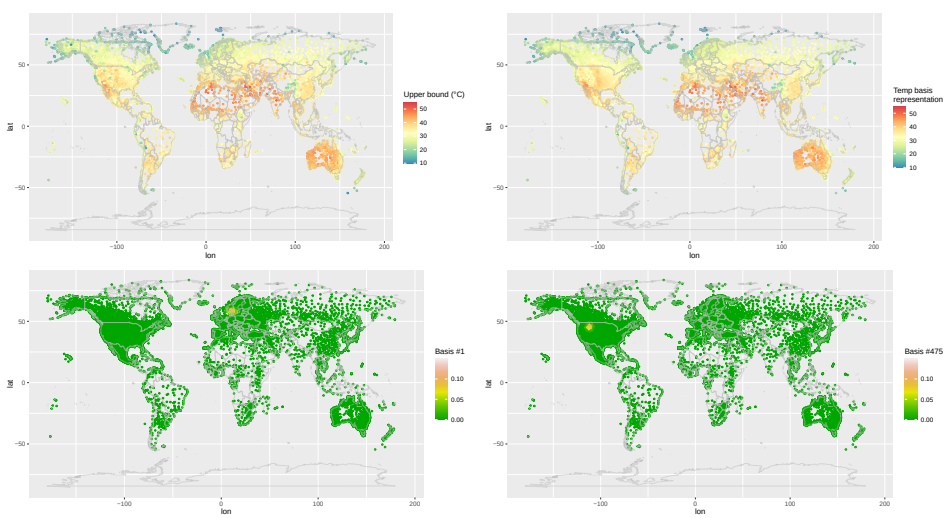

FIGURE 1. Temperature upper bound field, two example local basis functions, and a basis reconstruction using $X\hat{\boldsymbol{\beta}}_{\text{svd}}$ (top right).

(1) The bottom panels of Figure 1 displays basis #1 and basis #475, which correspond to the localized bumps near southern Sweden and northern Montana, respectively. Here is the code to visualize the

basis functions using `ggplot`:

```
library(ggplot2)
basis_num <- 1
ggplot(stations) + geom_point(aes(x=lon, y=lat, col=
    Xmat[,basis_num]),size=0.7) +
  geom_path(data = map_data("world"), aes(x = long, y = lat,
      group = group), color = "gray", linewidth = 0.2,
      alpha=0.5) +
  scale_color_gradientn(colours = terrain.colors(10), name =
  paste("Basis #", basis_num), na.value = "transparent",
  limits=c(0,0.15))
```

Choose any other two basis functions and visualize them.

(2) In 2–4 sentences, explain what a *local basis function* means in this context. Use Figure 1 and your visualization in (1) to support your explanation.

(3) Similarly to simple linear regression, the projection of $\mathbf{y}$ onto the linear space $\mathcal{S} = \text{span}(\mathbf{X})$ is obtained through

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}, \qquad \hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}.$$

(a) Run `solve(t(Xmat)%*%Xmat)` and report what happens (error message or warning). In one paragraph, explain what singular (or numerically singular) means in this context.

(b) Let singular value decomposition (SVD) of $\mathbf{X}$ be

$$\mathbf{X}_{n\times p} = \mathbf{U}_{n\times p}\mathbf{D}_{p\times p}\mathbf{V}^T_{p\times p}.$$

Here, the columns of $\mathbf{U}$ are the *eigenarrays*, $\mathbf{D}$ has singular values and is diagonal with $\sigma_1 \geq \cdots \geq \sigma_p \geq 0$. The SVD represents an expansion of the original basis functions in a coordinate system where the covariance matrix is diagonal. Plot the singular values of $\mathbf{X}$ using the following code:

```
s <- svd(Xmat, nu=0, nv=0)$d
plot(s, type="b", pch=20, xlab="index", ylab="singular
    value")
plot(log(s), type="b", pch=20, xlab="index",
    ylab="singular value in log scale")
```

Describe what you see. Comment on why tiny singular values would cause instability in inverse problems.

(4) Using SVD, compute the projection coefficients $\hat{\boldsymbol{\beta}}$ using the pseudoinverse form

$$\hat{\boldsymbol{\beta}}_{\text{svd}} = \mathbf{X}^{+}\mathbf{y} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{\top}y, \qquad \boldsymbol{\Sigma}^{+} = \operatorname{diag}\left(\frac{1}{\sigma_i}\right).$$

(a) Implement in `R` and replicate the top right panel of Figure 1 using $X\hat{\boldsymbol{\beta}}_{\text{svd}}$. Comment on the reconstruction quality. In particular, discuss what happens in the Rocky Mountain and Great Plains regions of the contiguous U.S. and why this approach can suppress small-scale noise from measurement error in the observational record.

(b) Also produce: (i) a histogram of coefficients $\hat{\boldsymbol{\beta}}_{\text{svd}}$, and (ii) a histogram of residuals $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_{\text{svd}}$.

(c) Explain why the coefficient histogram can show extreme outliers/huge magnitudes even if the reconstruction $X\hat{\boldsymbol{\beta}}_{\text{svd}}$ looks visually reasonable in Figure 1.

(5) In downstream modeling, $\boldsymbol{\beta}$ is sometimes assumed to follow a multivariate Gaussian distribution, and extremely large coefficients can break that assumption and destabilize inference. This is why it is undesirable if $\hat{\boldsymbol{\beta}}$ contains extremely large magnitudes (e.g., on the order of 10^{16}).

A common fix is *truncated SVD*: set $1/\sigma_i = 0$ when σ_i is below a threshold τ. Define

$$\hat{\boldsymbol{\beta}}_{\tau} = \mathbf{V}\operatorname{diag}\left(\frac{1}{\sigma_i}\mathbf{1}\{\sigma_i \geq \tau\}\right)\mathbf{U}^{\top}\mathbf{y}.$$

(a) Implement truncated SVD and compute coefficients for a threshold τ of your choice. Justify your choice. (I suggest you choose the threshold based on the figures you produced in (3b).)

(b) Compare the histograms of coefficients before and after truncation (i.e., $\hat{\boldsymbol{\beta}}_{\text{svd}}$ vs. $\hat{\boldsymbol{\beta}}_{\tau}$). Briefly interpret the difference.

(c) Produce another rescontruction figure using $\mathbf{X}\hat{\boldsymbol{\beta}}_{\tau}$

(6) **Bonus 10 points: Tikhonov (ridge) damping.**

Instead of truncation, consider ridge/Tikhonov regularization:

$$\hat{\boldsymbol{\beta}}_{\lambda} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2.$$

Using the SVD, show that

$$\hat{\boldsymbol{\beta}}_{\lambda} = \mathbf{V}\operatorname{diag}\left(\frac{\sigma_i}{\sigma_i^2 + \lambda}\right)\mathbf{U}^{\top}\mathbf{y}.$$

Implement this and compare (for a few λ values) RMSE and coefficient magnitude to truncated SVD.

Problem 3 (Simple simulation study; 10 points)**.** Consider estimating the shape (α) and scale (β) parameters of a Gamma(α, β) pdf based on a random sample of size n from the pdf. Use two approaches:

(1) maximum likelihood estimation;
(2) method of moments.

Compare the bias and MSE of these two estimators. Plot this for $n = 20$ to 1000 (you do not need to do this for every value, just a grid). Let $\alpha = 3$, $\beta = 7$ (with parameterization s.t. the expected value is 21).

Problem 4 (Monte Carlo methods for random matrices; 28 points)**.** Consider the following approach for generating random covariance matrices of dimension $m \times m$: Simulate m^2 $N(0, 1)$ random variables to obtain an $m \times m$ matrix R. Obtain the positive definite covariance matrix $\Sigma = RR^T$. Note that this is a way to simulate a matrix with a particular Wishart distribution. Read through all the parts below before beginning to work on this problem.

(1) What is the expected value of the trace(Σ) generated using the above approach (note that you need to obtain this value analytically using distribution theory)? Plot approximate density plots for the trace. Report Monte Carlo standard errors and the sample size you used in each case. Do this for $m = 100$ and $m = 1000$.
(2) What is the expected value of the largest eigenvalue of Σ? It might be hard to get this value anlytically. Simply report approximate density plots for the largest eigenvalue. Report Monte Carlo standard errors and the sample size you used in each case. Do this for $m = 100$ and $m = 1000$.
(3) Now study how the expected largest eigenvalue changes as a function of m.
    (a) Plot approximate $\mathrm{E}(\lambda_1)$, expected value of largest eigenvalue, versus m.
    (b) Report your grid of m values
    (c) What is the largest m value for which you approximated the expectation?
(4) Provide the computational complexity for the code you have written to approximate the expected value of the largest eigenvalue. Do this in terms of flops (floating point operations).

Problem 5 (Code optimization & parallel computing; 40 points)**.** If you are planning to share, package, or use your code repeatedly, you might consider the *efficiency* of your code. Running your code and *timing* it is a good starting point. On top of `system.time()`, another way to time code in `R` is

```
ptm <- proc.time()
# Your code chunk
proc.time() - ptm
# user system elapsed
# 0.34   0.06   0.41
```

The values presented (user, system, and elapsed) will be defined by your operating system, but generally, the *user* time relates to the execution of the code, the *system* time relates to system processes such as opening and closing files, and the *elapsed* time is the difference in times since you started the stopwatch (and will be equal to the sum of user and system times if the chunk of code was run altogether and single-threaded). Note you should not run the previous chunk of code line by line — you need to run the entire chunk as a whole to avoid idle user time.

For this problem, we will be examining the file `slow_code.R` from Canvas. Read this script and understand what `myfunc()` is trying to achieve. Then you have to complete the following tasks:

(1) Run the script on your computer and report the elapsed/running time to finish running;

(2) Perform serial optimization on the `myfunc()` function to decrease the elapsed/running time. You can use any techniques including the ones covered in class, e.g., using look-up tables, avoid power operation `^` or `sin()` etc., pay attention to stride.

Please report the values of the `Res` vector, and compare them to the ones from (1). If they are different, re-check your code and see if there are any bugs.

When the result is corrected, report the improved running time using the same computer that you used for (1). Note the team with the most relative improvement will receive 10 bonus points.

(3) Parallelize your optimized code using the `foreach` parallel back-end. Under **one** Hellbender node, request and register a number of cores and see how much more efficient the parallelization is compared to (2). Plot the running time versus the number of cores `n_cores`; you can choose a grid of `n_cores` values $\{4, 8, 16, 32, 64\}$.

Problem 6 (`python` exaple; 12 points)**.** Consider the data matrix at `Mat.dat`. You need to compute the means of the first and third columns. Note that reading in this matrix in `R` is computationally intensive because `R` uses column-major order and there are a large number of columns `Mat.dat`.

(1) Instead of directly reading the matrix into `R`, take the transpose of the matrix using `python`. Read the transposed matrix into R and carry out the necessary calculations in `R`;

(2) Now use `python` to do the entire calculation without reading the matrix into `R`. Show that you get the same results as in (1).

References

Mark D Risser, Likun Zhang, and Michael F Wehner. Data-driven upper bounds and event attribution for unprecedented heatwaves. *Weather and Climate Extremes*, 47:100743, 2025.