P. Koumoutsakos, P. Hadjidoukas
ETH Zentrum, HG D1.1/G3
CH-8092 Zürich

Set 05 - Inference on the linear model.

Issued: March 26, 2018
Hand in: April 9, 2018

Question 1: Maximum likelihood using Intel MKL

Intel® Math Kernel Library is a collection of highly optimized mathematical routines for the Intel architecture.

If you are using an Intel architecture on your platform, you can get a student licensed software called Intel Parallel Studio[1], which includes both libraries, as well as the powerful `icc` compiler. Otherwise, use Euler, and run before doing anything else: `module load new parallel_studio_xe/2018.0`.

In this exercise, we will build an artificial data set using a vector $\beta \in \mathbb{R}^p$ of our choosing. We will generate N random points $x_i \in \mathbb{R}^p$ and calculate y_i by perturbing the linear relationship with Gaussian noise $\varepsilon_i \sim \mathcal{N}(0, 1)$:

$$y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i. \tag{1}$$

We have already seen that the maximum likelihood estimate of $\boldsymbol{\beta}$ is the least squares solution:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \tag{2}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}. \tag{3}$$

While mathematically the problem is solved, it requires some computational care. Direct solving might cause issues, see for example the Läuchli matrix in [2, p. 239]. We instead use methods which exploit the structure of the problem beyond merely treating it as a general system of linear equations.

a) QR decomposition

Any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}, m > n$ can be decomposed in the following way: [1, p. 82]:

$$\mathbf{A} = \mathbf{QR}, \ \mathbf{Q} \in \mathbb{R}^{m \times n}, \ \mathbf{R} \in \mathbb{R}^{n \times n}, \tag{4}$$

$$\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}, \tag{5}$$

and $\mathbf{R}$ is an upper triangular matrix.

i) Show that $\hat{\boldsymbol{\beta}} = \mathbf{R}^{-1} \mathbf{Q}^\top \mathbf{y}$

[1] https://software.intel.com/en-us/qualify-for-free-software/student

ii) Consult the documentation for Intel MKL and list the necessary routines. Note that you should not explicitly store $\mathbf{R}^{-1}$ or $\mathbf{Q}^{\top}$.

iii) Implement the method using the MKL routines. Run multiple experiments:
- Validate your kernel by comparing it to the direct least squares solver `LAPACKE_dgels`. How does the error $\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|$ behave, $\hat{\boldsymbol{\beta}}$ the LS estimate and $\boldsymbol{\beta}$ the true parameter? Let $N = 2^1, \dots, 2^{15}$.
- Measure the computational runtime $N = 2^1, \dots 2^K$. Set K initially to 15 but try increasing it to see how high you can go, before experiencing any issues, if any.
- Investigate if the runtime decreases if you allow multiple threads – that is, investigate what options exist for multi-threading.

Pick $p = 30$.

Question 2: Tasking on Markov Trees

A **Markov chain** $\{x_k\}_{k \subset \mathbb{N}}$ is a collection of random variables x for which the following property holds:

$$\mathbb{P}\left[x_{k+1} \mid x_1 \dots x_k\right] = \mathbb{P}\left[x_{k+1} \mid x_k\right]. \tag{6}$$

The quantity $\mathbb{P}\left[x_{k+1} \mid x_k\right]$ is called the **transition probability**. Intuitively the above property means that the probability of the the next state depends only on the previous state, but no further history. The index k plays the role of discrete time.

In this exercise we focus on discrete Markov chains, where x_k takes values from a finite set $\{1, \dots, M\}$. Such a Markov chain can be characterized by values $\mathbb{P}\left[x_{k+1} = j \mid x_k = i\right]$, which are usually stored in a matrix p_{ij}.

We define a tree as an acyclic graph for which the following properties hold:

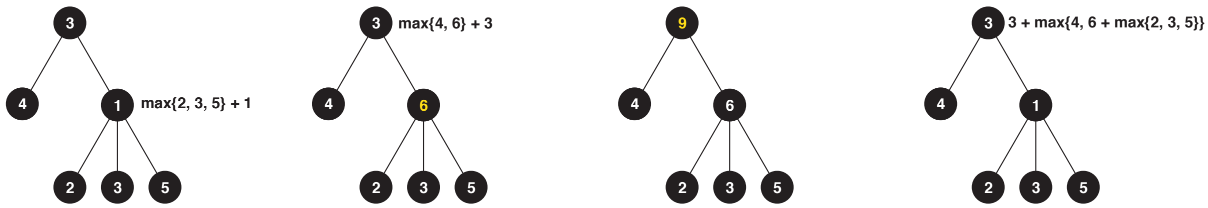
- A node can be a **child**, **parent**, or a **sibling** (non-exclusive properties).
- A node can have at most one parent. Nodes sharing the same parent are said to be children of that parent, and each other's siblings.
- There is exactly one node with no parent, called the **root** node.
- Each node in the graph has a value, an independent random variable $\sim$ Exp(λ).
- The number of children a node has depends on the number of the children its parent has in a Markovian sense:

$$\texttt{p[i][j]} = \mathbb{P}\left[\#\text{children(node)} = j \mid \#\text{children(parent(node))} = i\right]. \tag{7}$$

We are interested in generating such a random graph and finding the path from the root to a descendant that maximizes the sum of nodal values. We impose a limit of maximum two children per node.

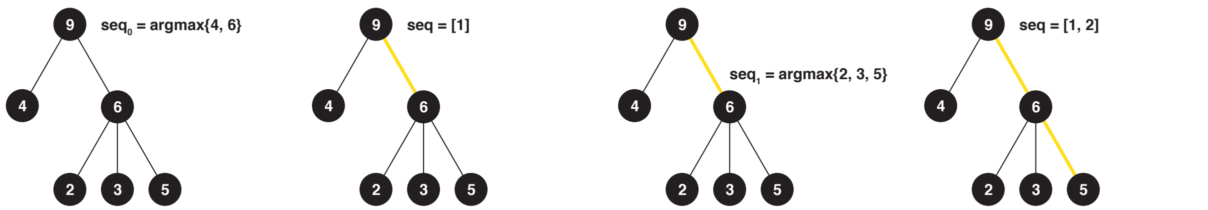
a) Implement the functions `init_markov_root`, `sample_num_children`, and `init_markov_tree`. The `init_markov_tree` uses `sample_num_children` to recursively generate the nodes. `init_markov_root` initializes the first node in the tree and assigns it its random value.

   Sampling X from set $\{1, \dots, M\}$ with probabilities $\mathbb{P}\left[X = i\right] = p_i$ can be done as follows:



(a) Step 1 (b) Step 2 (c) End (d) Recursive formulation

Figure 1: First (backwards) pass: finding the maximum path-sum



(a) Step 1 (b) State after step 1 (c) Step 2 (d) Final state

Figure 2: Second (forward) pass: finding the optimal path

1. Generate $U \sim$ Unif$(0, 1)$.
2. Find j such that $\sum_{k=1}^{j} p_k \leq U < \sum_{k=1}^{j+1} p_k$
3. Set $X = j$.

b) Implement the algorithm for the maximum nodal sum and path that reaches it. Use the two pass dynamic programming algorithm displayed in figures 1 2.

   Complete the functions `max_sum_pass1` and `max_sum_pass2`.

c) Parallelize the routines using appropriate OpenMP constructs.

Consult the skeleton header files for a more detailed description of individual tasks. Node construction and destruction has already been implemented in `src/node.cpp`. The described algorithm is a simplification of the Viterbi algorithm, used in noisy signal recovery.

References

[1] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics (Texts in Applied Mathematics)*. Springer, 2010. ISBN: 978-3-642-07101-0.

[2] Josef Stoer. *Introduction to Numerical Analysis*. New York: Springer, 2002. ISBN: 978-1-4419-3006-4.