Latent Dirichlet Allocation

How to Write Fast Numerical Code - One to One #1

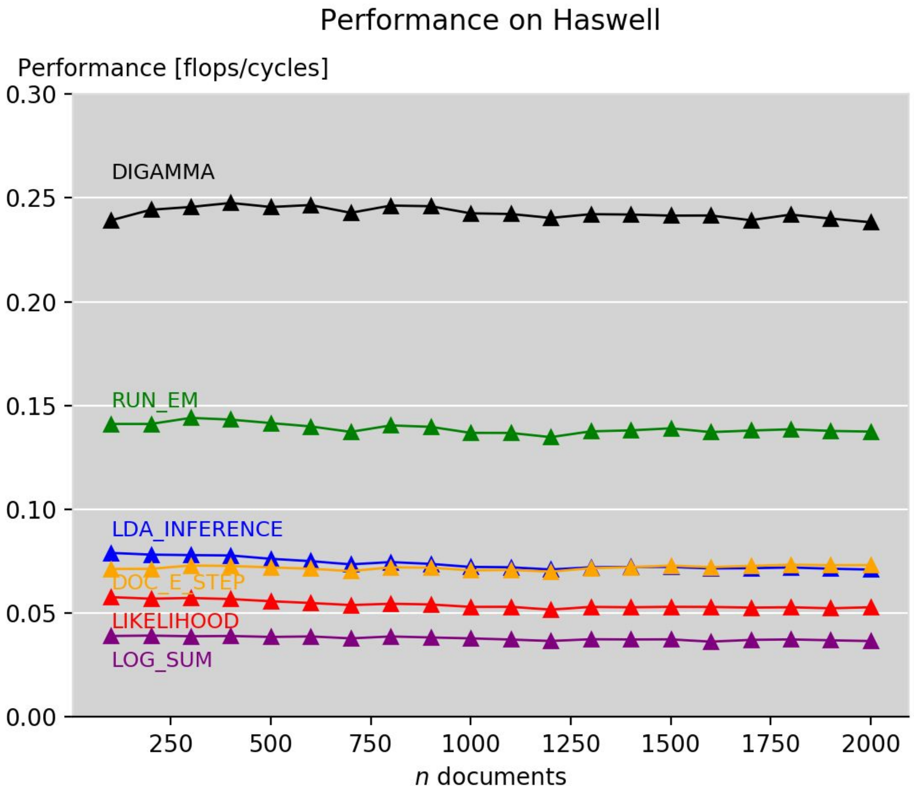
Description of the algorithm

```
Infer(Document d, $\alpha$, $\beta$):
  Do:
    # Update word-topic associations
    For n from 1 to $N_d$:
      For i from 1 to K:
        # See below for definition of $\mathbf{\Psi}$
        $\phi_{n,i}^{(new)} \leftarrow \beta_{i,w_n} \times \exp\left(\mathbf{\Psi}\left(\gamma_i^{(old)}\right) - \mathbf{\Psi}\left(\sum_j^K \gamma_j^{(old)}\right)\right)$
      Normalize($\phi_n^{(new)}$)

    # Update topic mixture (here all topics at once)
    $\gamma^{(new)} \leftarrow \alpha + \sum_n^N \phi_n^{(new)}$
  Until convergence
  Return $\phi, \gamma$

Learn(Corpus c):
  Do:
    # Expectation step
    For document d in c:
      $\phi^{(d)}, \gamma^{(d)} \leftarrow Infer\left(d, \alpha^{(old)}, \beta^{(old)}\right)$
    # Maximization step
    $\beta^{(new)} \leftarrow \sum_d \sum_n \phi_{n,i}^{(d)} w_{d,n}$
    $\alpha^{(new)} \leftarrow Newton\text{-}Raphson\left(\alpha^{(old)}\right)$ #Linear time due to special struct.
  Until convergence
  Return $\alpha, \beta$
```

Performance plot0



Validation

- We use a python script which is able to:
    - Generate results from our reference implementation for a given number of topics and documents.
    - Run the optimized implementation and compare its results against the reference results.
- The range of inputs we are considering is:
    - Number of documents: between 100 and 2000
    - Number of topics: between 10 and 50

Instrumentation

- Decided to run all the benchmarking on the lab machines where all the timing infrastructure is already set and we obtain the same results across the team
- rdtsc counter.
- For the timing infrastructure we had two requirements:
    - record nested timings;
    - be able to measure the runtime of a function that is called a non-deterministic number of times.
- timer structure which is instantiated for each function/code block we want to measure
- For non-deterministic number of iterations; we take the mean over performed iterations

Optimizations work plan

- Split the optimizations into
    - Vectorization (Frédéric, Saurav)
    - ILP, scalar replacement, blocking etc (Cristina, Benjamin)
- Vectorization
    - Vectorize all the mathematical standalone functions first
    - Bottom-up approach to vectorize the rest of the code
- Other functions: start with the most-called functions (lda_inference, etc.)
    - Blocking (frequently going over a matrix to gather some counts)
    - Data representation: already good
    - Things to check for lda_inference, doc_e_step, likelihood
        - Check unit stride and TLB misses for the matrices accesses in.
        - Unroll lda_infoerence, doc_e_step, likelihood for ILP
        - Register spills minimization given the large amount of variables used in each of them