Maximum-Likelihood Estimation
With Newton-Raphson Iteration

Vincent Kang Fu

CS 4961

December 2010
Background

- Newton-Raphson iteration for maximum-likelihood estimation is implemented in many professional statistical packages.

- Can I write my own version in C that is faster than these highly optimized statistical packages?
Maximum Likelihood Estimation

- Prominent method for estimating statistical models
- Choose model coefficients to maximize the joint likelihood of the observed outcomes
- Statistical estimation becomes an optimization problem
- Use Newton-Raphson iteration for optimization
Newton-Raphson Iteration

- General method for optimizing a function
- Set the first derivative to zero and apply Newton's method to solve
- Iterative procedure:
  - $\beta_0 = \text{starting value}$
  - for $i = 0, 1, \ldots, \infty$ until convergence do
    $$\beta_{i+1} = \beta_i - \frac{\partial^2 l}{\partial \beta^2}^{-1} \frac{\partial l}{\partial \beta}$$
Modeling Married Women's Labor Force Participation

- $N = 1,984,591$ married women from the 1990 US Census
- Model: Binary logit model

$$p_i = \text{labor force participation probability}$$

$$\log \frac{p_i}{(1 - p_i)} = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots$$
Data Arrays

- $Y$ is an $N \times 1$ vector where $y_i = 1$ for women in the labor force, 0 otherwise.
- $X$ is an $N \times K$ array of covariates including race, education, age, other income in the household, household size, presence of young children.
Maximum Likelihood Estimation

- The *likelihood* \( L \) is the joint probability of the observed outcomes
- Find coefficients \( \beta \) to maximize \( L \)

\[
L(\beta) = \prod_{y_i=1} p_i \prod_{y_i=0} (1 - p_i), \quad p_i = \frac{\exp(\beta' x_i)}{1 + \exp(\beta' x_i)}
\]

- Easier to maximize \( l \), the log likelihood
- Need the first and second derivatives for Newton-Raphson iteration
First Derivative

\[ \frac{\partial l}{\partial \beta} = \begin{bmatrix} \sum (y_i - p_i) x_{0i} \\ \sum (y_i - p_i) x_{1i} \\ \vdots \\ \sum (y_i - p_i) x_{Ki} \end{bmatrix} \]

for (i = 0; i < N; i++)
  for (k = 0; k < REGRESSORS; k++)
    result[k] += (y[i] - p[i]) * x[i][k];

- Calculating elements in parallel sacrifices locality in access to X array
Second Derivative

\[
\frac{\partial^2 l}{\partial \beta^2} = -\begin{bmatrix}
\sum p_i (1-p_i) x_{0i}^2 & \sum p_i (1-p_i) x_{0i} x_{1i} & \ldots & \sum p_i (1-p_i) x_{0i} x_{Ki} \\
\sum p_i (1-p_i) x_{1i} x_{0i} & \sum p_i (1-p_i) x_{1i}^2 & \ldots & \sum p_i (1-p_i) x_{1i} x_{Ki} \\
\vdots & \vdots & \ddots & \vdots \\
\sum p_i (1-p_i) x_{Ki} x_{0i} & \sum p_i (1-p_i) x_{Ki} x_{1i} & \ldots & \sum p_i (1-p_i) x_{Ki}^2
\end{bmatrix}
\]

for(r = 0; r < REGRESSORS; r++)
    for(i = 0; i < N; i++)
        for(c = 0; c < REGRESSORS; c++)
            result[r][c] += p[i]*(1-p[i])*x[i][r]*x[i][c];
Optimization

- Take advantage of locality in looping
- Openmp
  - Task parallelism: calculate first and second derivatives in parallel
    \[ \beta_{i+1} = \beta_i - \frac{\partial^2 l}{\partial \beta^2} \left( \frac{1}{\partial l} \frac{\partial l}{\partial \beta} \right) \]
  - Openmp reduction clause cannot operate on array targets
- SSE3 Vector optimization
Because inverting a matrix is costly, use Intel MKL's `LAPACKE_dgesv` to solve the linear system for the $\delta$ vector in the update step of each Newton-Raphson iteration.

\[
\begin{align*}
\frac{\partial^2 l}{\partial \beta^2} \begin{bmatrix}
\delta_0 \\
\delta_1 \\
\vdots \\
\delta_K
\end{bmatrix} &= \frac{\partial l}{\partial \beta} \\
\beta_{i+1} &= \beta_i - \delta
\end{align*}
\]
Platform

- Intel Core i3-530 2 cores, 4 threads
  - Private 32K + 32K L1 cache
  - Private 256K L2 cache
  - Shared 4MB L3 cache
- Ubuntu Linux 10.10 32-bit edition
  - Intel icc 11.1.073
  - Intel MKL 10.3.1.107 (linear algebra)
- Windows XP SP3
  - Stata/SE 10.1
Execution Time by Platform

<table>
<thead>
<tr>
<th>Platform</th>
<th>Execution Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stata/SE 10.1</td>
<td>15</td>
</tr>
<tr>
<td>Naïve sequential</td>
<td>27</td>
</tr>
<tr>
<td>Locality</td>
<td>3.2</td>
</tr>
<tr>
<td>Locality + Openmp</td>
<td>2.1</td>
</tr>
<tr>
<td>Locality + SSE3</td>
<td>3.2</td>
</tr>
<tr>
<td>Locality + Openmp + SSE3</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Discussion

- My implementation was faster!
- Great benefit from locality optimization
- Small improvement from openmp
- No benefit from SSE3
- Suggestions welcome!