Non-Linear Least Square

Monika Bhole – 3640920 – st185128@stud.uni-stuttgart.de
Hrishikesh Kate – 3643558 – st184581@stud.uni-stuttgart.de

1. Introduction

The non-linear least-squares approach is used to optimize the parameters of a non-linear function, which helps to create a fair prediction between the independent variables and the set of observations. NLS has to minimize the difference between the real data points, and the values that are generated by the non-linear model itself. This systematic difference is usually evaluated by a sum of squared residuals, where residuals mean the computed numbers that represent the relationships between observed and predicted measurements. NLS focuses on the convergence of the objective function to a value that results in minimized parameters.

Consider a model function *y= f(x, a)* where *x* is an independent variable and *a* is set of parameters. This function illustrates a procedure whose outcomes ŷ we already know.
Residuals: are the difference between observed and predicted values. The subscript i refers to the data point.

$$r_i = y_i - \hat{y}_i$$

Sum of squared errors is given by squaring and summing the residuals.

$$S = \sum_i^N \left(y_i - \hat{y}_i\right)^2 = \sum_i^N r_i^2$$

Finding a set of parameters, '*a*', that produce a 'y' outcome that closely resembles ŷ is the main goal. One method to measure how close we are to ŷ involves computing the sum of squared residuals. We produce the lowest possible squared error by varying the values of the parameters. For the function, this parameter *a* will be the best fit. To determine the minimum value, a widely used approach is to compute its derivative with respect to a certain variable and setting it to zero. In this instance, we wish to find a number that minimizes the function S. The subscript j is for multiple values of *a,* as the function depends on one or more parameters.

$$\frac{\partial S}{\partial a_j} = \frac{\partial}{\partial a_j} r_i^2 = \frac{\partial}{\partial a_j} \sum_i^N \left[f\left(x_i, a_j\right) - \hat{y}_i\right]^2 = 0$$

This equation can be represented in matrics form as:

$$2J^T\left[f\left(x_i, a_j\right) - \hat{y}_i\right] = 0$$

In order to solve this equation and find the parameters that best fits the function we can use three algorithms.

1. Gradient Descent

Gradient descent is an optimization approach that identifies local minimum of a function. The function we're attempting to minimize is differentiable, so we calculate the gradient at any point. This implies we know the direction we need to go to continue heading down. With each iteration, we get a little closer to the function's minimum. The gradient descent approach has two crucial aspects: the initial guess and the size of the steps (learning rate) we take at each iteration. The effectiveness of this strategy is heavily dependent on these two factors. The gradient descent step is represented as: $h_{GD} = \alpha J^T r$

2. Gauss-Newton

Gauss Newton is somewhat similar to Gradient descent, which involves iterative stages to get an optimal solution. Here, this method deals specifically with subduing the sum of the squared residuals in a non-linear regression problem. Linearization of the model around the current estimated parameters is performed in each iteration. In Gauss-Newton, the parameter is updated by: $a_{n+1} = a_n + h_{GN}$ which is iterated, and the step is calculated using: $h_{GN} = inverse(J^T J)J^T[\hat{y} - f(a_n)]$. This gradual approach to find optimal settings for the parameters is termed convergence, where the curve-fitting application can take various variables and form complex relationships between them by utilizing them.

3. Levenberg-Marquardt

The Gauss-Newton and gradient descent methods are combined to create the Levenberg-Marquardt algorithm. The damping factor is denoted by λ, and I stand for an identity matrix. This Method uses a Gauss-Newton step when λ is small and the gradient descent method when λ is big.

The equation can be expressed as: $(J^T J + \lambda I)\, h_{LM} = J^T[\hat{y} - f(a_n)]$

2. Objective

The objective of this project is to implement a non-linear least square using an efficient algorithm on GPU as well as CPU. The tasks performed during the development of this project are as follows:

A. Implementation of Non-Linear Least Square on CPU using Gauss Newton algorithm.
B. Implementation of Non-Linear Least Square on GPU using Gauss Newton Algorithm.
C. Comparison between the speed-up of GPU and CPU for both the algorithms.

3. Implementation steps for Gauss-Newton:

1. Initializing the Parameters: Here, we start by setting the parameters of the non-linear model to some values.
2. Defining the Loss/Cost Function: The cost function, is usually the sum of squares of residuals, that measures the gap of observed data and model forecasts.
3. Computing the Jacobian matrix: Determine the first-order partial derivatives of the model function for each parameter by computing the Jacobian matrix. For every data point, this includes calculating the derivatives of the model function.
4. Compute Residuals: Check the residuals by subtracting the observations from the model results.
5. Updating the parameters accordingly: $a_{n+1} = a_n + h_{GN}$
6. h_{GN} is given by, $h_{GN} = inverse(J^T J\ )J^T[\hat{y}\ \ - f(a_n)]$
7. Check Convergence
8. Finalizing the parameters: The goal is to reach convergence, after which the final estimated parameters comprise the best solution for the nonlinear regression problem.

Note: In Implementation, the Eigen library is used for matrix and vector operations in the calculation of the Jacobian matrix and the residual vector.

4. Contribution of team members

Team member	Task
Monika Bhole	• Literature review • Finalization of algorithm • GPU and CPU implementation • Testing and Debugging • Report writing
Hrishikesh Kate	• Literature review • CPU and GPU implementation • Testing and Debugging • Code integration • Analysis and review of results

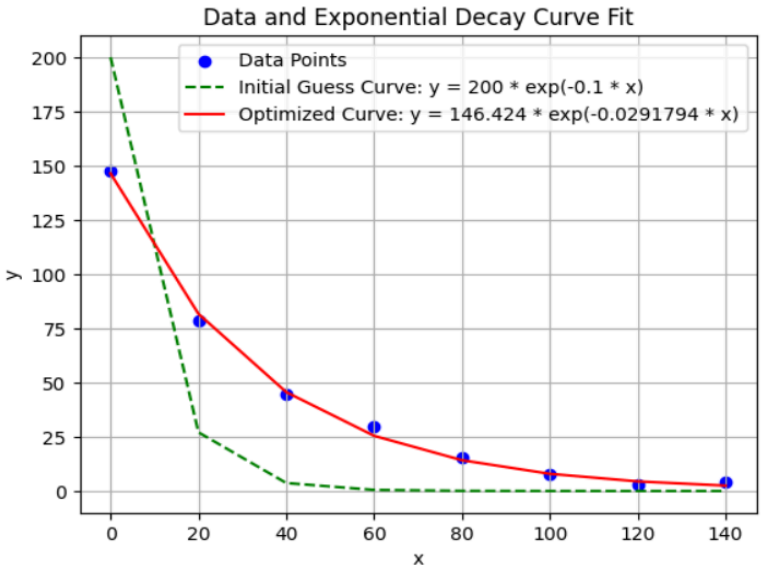
5. Results and Discussion

In this section, an example is proposed using the Gauss-Newton method. Consider the below data points (t_i, y_i).

Table 1: Observations

Time (t)	0	20	40	60	80	100	120	140
Drug (y)	147.8	78.3	44.7	29.5	15.2	7.8	3.2	3.9

The function is given by: ***y = a exp(-kt) This*** function is considered for the observations in Table 1. If we take the initial guess as a = 200 and k=0.1, using CPU and GPU implementation, after 11 iterations we get parameter values as a = 146.424, k = 0.0291794 and model as: ***y = 146.424 exp(-0.0291794t).*** The optimized curve closely follows the actual data points, indicating a good fit.



The residuals for each data point from 0 to 7 are obtained for 11 iterations as:

Table 2: Comparative Residuals for CPU and GPU Implementations

Iteration	**CPU Residuals**	**GPU Residuals**
1	[-52.2, 51.2329, ..., 3.19877, 3.89983]	[-52.2, 51.2329, ..., 3.19877, 3.89983]

2	[0.582403, -153.624, ..., -2247.28, -3541.47]	[0.582403, -153.624, ..., -2247.28, -3541.47]
...	...	...
11	[1.3755, -3.38922, ..., -1.21489, 1.43696]	[1.3755, -3.38922, ..., -1.21489, 1.43696]

The SSE values for 11 iterations are: SSE= [8190.07,2.08304E+07,235524,49345.1,13024.6,872.77,35.3424,35.33,35.33,35.33,35.33]

After an initial period, the SSE values stabilize from iteration 7 onwards, which shows the model parameters are converging towards values that offer an optimal fit to the observed data. After 7 iterations there is minimal change in SSE which suggests that the model has reached a point where further iterations do not significantly alter the fit, this indicates correct parameter estimation using Gauss-Newton.

6. Performance Evaluation

CPU: Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz

GPU: NVIDIA Corporation GK104 [GeForce GTX 680] (rev a1)

Host name: kale.cis.iti.uni-stuttgart.de(bholema)

CPU Implementation:

- Convergence was achieved in 11 iterations.
- The estimated parameters were a_cpu = 146.424 and k_cpu = 0.0291794.
- The total execution time was 0.000145 seconds.

GPU Implementation:

- The algorithm also converged in 11 iterations when executed on the GPU.
- The estimated parameters on the GPU were a_gpu = 146.424 and k_gpu = 0.0291794, matching those computed by the CPU implementation.
- The GPU execution time, including memory copy, was 0.000016 seconds, with the memory copy time accounting for 0.000002 seconds of this total. The execution time excluding memory copy operations was 0.000014 seconds.
- The speedup achieved by the GPU implementation over the CPU, excluding memory copy time, was approximately 10.3571 times. Including memory copy times, the speedup was 9.0625 times.

```
bholema@kale:~ > lspci | grep VGA
03:00.0 VGA compatible controller: NVIDIA Corporation GK104 [GeForce GTX 680] (rev a1)
04:00.0 VGA compatible controller: NVIDIA Corporation GK104 [GeForce GTX 680] (rev a1)
0b:00.0 VGA compatible controller: Matrox Electronics Systems Ltd. G200eR2
83:00.0 VGA compatible controller: NVIDIA Corporation GK104 [GeForce GTX 680] (rev a1)
84:00.0 VGA compatible controller: NVIDIA Corporation GK104 [GeForce GTX 680] (rev a1)
```

```
bholema@kale:~ > grep name /proc/cpuinfo
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
model name      : Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
```

```
NonLinearLeastSquare | NonLinearLeastSquare | NonLinearLeastSquare | NonLinearLeastSquare | NonLinearLeastSquare | NonLinearLeastSquare | NonLin
/zhome/bholema/Downloads/NonLinearLeastSquare/build> /zhome/bholema/Downloads/NonLinearLeastSquare/build/NonLinearLeastSquar
CPU Converged in 11 iterations.
Estimated parameters:
a_cpu = 146.424, k_cpu = 0.0291794
CPU Time: 0.000145s
Context has 4 devices
Running on NVIDIA GeForce GTX 680 (3.0)
GPU Converged in 11 iterations.
Updated parameters gpu: a = 146.424, k = 0.0291794
GPU Time 0.000016s
Updated parameters gpu: a_gpu = 146.424, k_gpu = 0.0291794
Memory copy Time: 0.000002s
GPU Time w/o memory copy: 0.000014s (speedup = 10.3571)
GPU Time with memory copy: 0.000016s (speedup = 9.0625)
*** Finished ***
```

7. Conclusion

These results display computational efficiency that can be achieved through parallel processing on GPUs for NLLS problems. The GPU implementation, which uses NVIDIA GeForce GTX 680, gave a significant reduction in computation time, even when considering the memory copy operations to execute the same task. Hence, the GPU is significantly faster than that of the CPU.

8. References

[1] Teunissen, Peter. 1990. "Nonlinear Least Squares." *Espace.curtin.edu.au*. http://hdl.handle.net/20.500.11937/38758.

[2] "Least Squares." 2019. Wikipedia. December 19, 2019. https://en.wikipedia.org/wiki/Least_squares .

[3] Manfre, Diego. 2021. "The Interesting World of Non-Linear Regressions." Medium. February 5, 2021. https://towardsdatascience.com/the-interesting-world-of-non-linear-regressions-eb0c405fdc97.

[4] Lai, Wen, Sie Kek, and Kim Tay. 2017. "Solving Nonlinear Least Squares Problem Using Gauss-Newton Method." *IJISET -International Journal of Innovative Science, Engineering & Technology* 4 (1): 2348–7968. https://ijiset.com/vol4/v4s1/IJISET_V4_I01_35.pdf .

[5] Wikipedia Contributors. 2021. "Gauss–Newton Algorithm." Wikipedia. Wikimedia Foundation. May 27, 2021. https://en.wikipedia.org/wiki/Gauss%E2%80%93Newton_algorithm .

[6] Hao, Wenrui. 2021. "A Gradient Descent Method for Solving a System of Nonlinear Equations." *Applied Mathematics Letters* 112 (February): 106739. https://doi.org/10.1016/j.aml.2020.106739 .