Parameter Extraction from Least Square Fitting

Yazhi Fan (yf92) and Yijia Chen (yc2366)

Program Organization

1. parameterExtraction.cpp / .h

ParameterExtraction.cpp contains parameter extraction methods (quasi-Newton and secant) and related helper functions. We followed the top bottom programming practice by setting up our parameter extraction method to be a function that takes a set of initial parameters (in a vector), a set of measurements (in a matrix), and a function that describes the model. So this method is not limited by the number of paratermers, variables, or type of function. Then we wrote smaller functions such as partialDerivative and getDelta to help achieve this goal.

2. EKV.cpp / .h

EKV.cpp contains the EKV model function, the normalized EKV model function, full grid search on initial guesses, and a function that reads text data file and construct variables measurement matrix.

3. test.cpp / .h

Test.cpp contains our validations and tests. There are two major tests: validation for power law model and asymptotic testing for EKV model. Validation for power law was done by randomly generating variables with known parameters and then pass those variables to the parameter extraction method. The result is compared with ground truth and percent error is reported. Asymptotic testing for EKV model was done by comparing IDmodel's values with the ID exponential function or quadratic function depending on if Vgs is less or greater than Vth. The relative errors are compared with a maximum allowed error (set to 0.5). An message with the result and relative error values is printed.

4. library

We used the the open source library Eigen to do all full matrix performances (such as inverse, multiply, and solve) in. Sice Eigen:Matrix is much more convenient to use than std::Matrix, we also used them in places such as storing variables' measurement data. We originally wrote a full matrix solver, but its functionality is limited. Implementing a method to find the inverse of any sized matrix seems too troublesome, so we used this library instead. Then it seemed redundant to have two matrix implementations, so we removed the matrix solver that we originally implemented.

5. Global constants (inputs)

There are many constants used in this program, such as threshold values, and they are defined in header files. The perturbation used in finite difference derivative approximation is set to 0.0001. Tolerance for quadratic convergence check is set to 0.2 (or 20%). Threshold on delta x norm to confirm convergence is set to 1E-7. Max number of iterations allowed is set to 1E7. Threshold value for V is set to 0.1 V. Max relative error allowed in asymptotic validation is set to 0.5.

Validation of parameter extraction program for power law (Task 2):

We first tried generating both random variable (x and y) internally. However, the parameter extraction performed very poorly because internally randomly generated x values do not cover a very good range to represent function behavior (for example, x = 1 is very useful for extraction of a good c0 but 1 is not likely to be randomly generated). So we instead randomly hand picked 10 values for x that covers a good range and had successful performance with several initial guesses. Our quasi-Newton parameter extraction performed well with power law models. For example, when our initial guess is c0 = 15.0 and m = -0.6, the result is shown below:

```
norm of delta x converged after 45980 iterations
parameter a[0] = 9.26652
parameter a[1] = -0.478951
V = 0.0270512
Power Law Validation Result:
||error||2 = 0.0845704
```

Validation on asymptotic behavior (Task 7):

To reduce user interaction with the EKV model, we implemented asymptotic validation checks. The validation generates three error calculations: exponential behavior error, quadratic behavior error, and insensitivity to Vds error. The relative errors are calculated by mean difference squared. Validation result report the error and specifies if each error is within a max allowed value for error.

Automated Convergence Check (Task 4):

We implemented verification for quadratic convergence by checking if the current ||V||2 is quadratically less than the previous ||V||2 within a percent tolerance. Convergence behavior on each iteration is mostly unpredictable in our program, and we rarely observed any quadratic convergence with both quasi-Newton method and secant method. This probably also relates to our program's slow convergence rate with the EKV model.
In addition, one of the stop condition we implemented is related to ||V||2: when current ||V||2 is greater than previous ||V||2, divergence occurred and parameter extraction failed.

Stop Conditions

1. ||Delta x|| threshold:
When delta x norm is below a specified threshold (set to 10E-7), convergence has occurred and program exits with current parameters.

2. Acceptable ||V||2 when appropriate ||delta x||2 threshold is difficult to find:
We hoped to base our stop condition on delta x norm. However, the optimal threshold is different for different problems and they are difficult to find. So we added this condition which declares convergence has finished when ||V||2 has stopped decreasing and ||V||2 is below an "acceptable" value. Note that ||V||2 can go beyond the "acceptable" value as long as it is still converging. This value only serves as a safe check.

3. Diverging case:
If ||V||2 is not below the acceptable value, and ||V||2 is increasing, then the result is diverging. Program exits with current parameters but also a warning message that says parameter search failed.

4. Max number of iterations:
If max number of iterations has been reached without convergence, convergence has failed and program exits with current parameters but also a warning message that says parameter search failed. This is implemented to prevent infinite or extremely long iterations.

Results and Observations

(All plots are created in MATLAB)
1. Downloaded measurements of ID, VGS, VDS (Task 3)

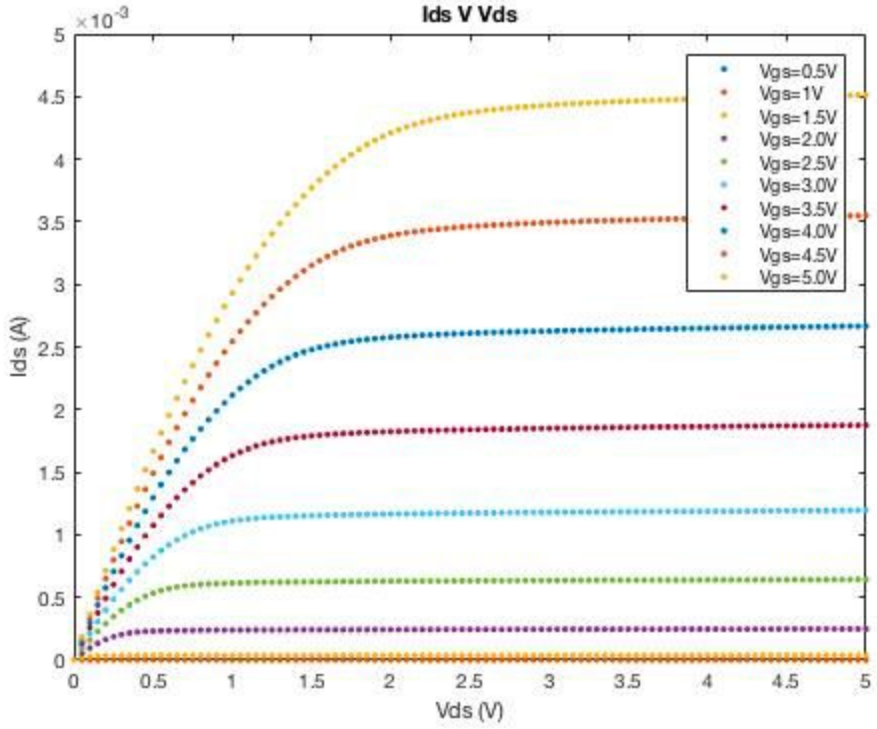


Figure 1. ID V Vds for IDmeasured.

2. EKV model with quasi-Newton and secant method (task 4)

Our quasi-Newton parameter extraction converged with the following result:

```
norm of delta x converged after 44217 iterations
parameter a[0] = 1.58029e-06
parameter a[1] = 0.571274
parameter a[2] = 0.2533
||V||2 = 4.20473e-05
||delta(x)||2: 0.000902937
```

Our secant method failed to converge. This may be a result of bad initial guesses.

```
PARAMETER SEARCH FAILED
Result is diverging. Initial guess may not be in the basin of
attraction.
parameter a[0] = 9.99997e-07
parameter a[1] = 0.999995
parameter a[2] = 0.999984
||V||2 = 0.000196084
||delta(x)||2: 4.23033
```

3. Normalized EKV (task 5)

We calculate normalized ID from $I_D(V_{GS}, V_{DS}; I_S, k, V_{th})/I_{Dmeasured}$ and plotted in MATLAB.

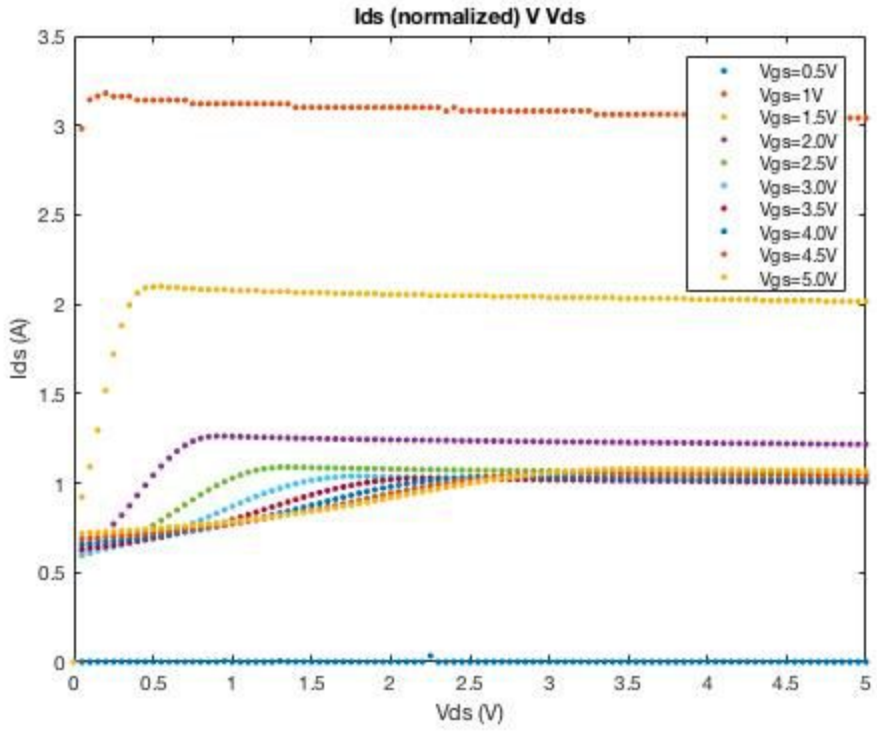


Figure 2. Normalized EKV Ids V. Vds

4. Full Grid Search

Among the 4 methods that we have implemented so far, only quasiNewton with non-normalized EKV performed well for a large range of initial guesses (although it took a long time and amount of iterations, but it did successfully converged in the end).

However, secant method and normalized EKV are hardly converging, so we tried the full grid search. Because convergence is slow, the full grid search would probably takes hours to run, so we only tried our best to pick a few values to search. In the search process, quasi-Newton for normalized EKV performed better than secant method. And quasi-Newton for normalized EKV performed the worst. Both of our secant methods failed the entire full grid search. They will only converges if we put in the rounded final answer (got from quasi-Newton method). The advantage of secant method is that it is cheap and fast, but it is too sensitive to initial guess.

	I_S	k	V_{th}	||V||	||delta x||
Task 4 (quasi Newton)	1.58029e-06	0.571274	0.2533	4.20473e-05	0.000902937
Task 4 (secant)	1.58029e-06	0.571274	0.2533	4.20473e-05	0.000902937
Task 5 (quasi Newton)	8.81137e-07	0.429298	1.35112	19.4955	1.18459e-05
Task 5 (secant)	8.81137e-07	0.429298	1.35112	19.4955	1.18459e-05

5. Visualization of result

We used the result from quasi-Newton method for this section since it has the smallest ||V||2.

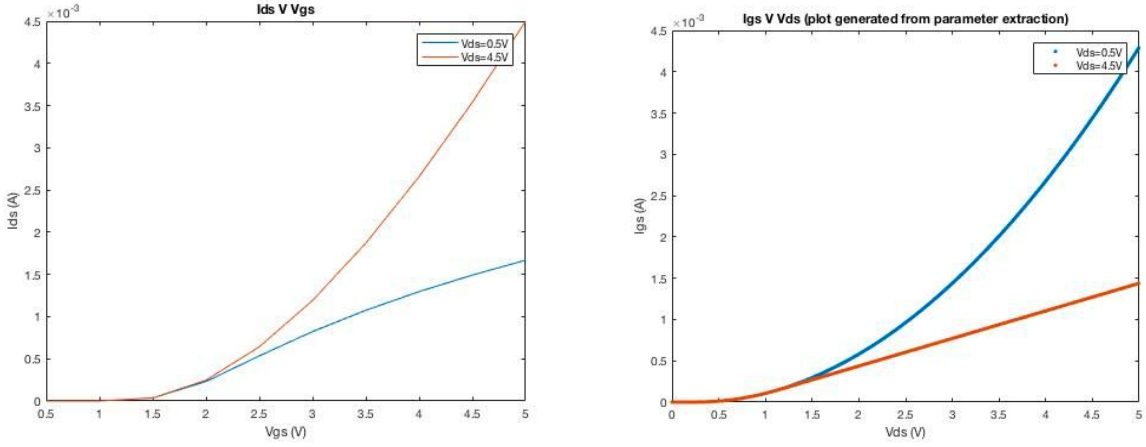


Figure 3. Ids V. Vgs before taking log. The left plot is IDmeasured, the right plot is IDmodel.

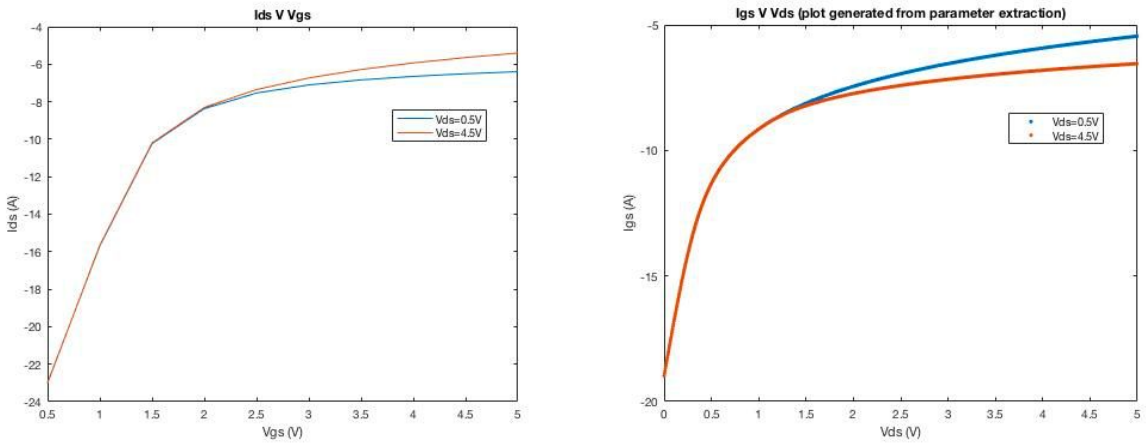
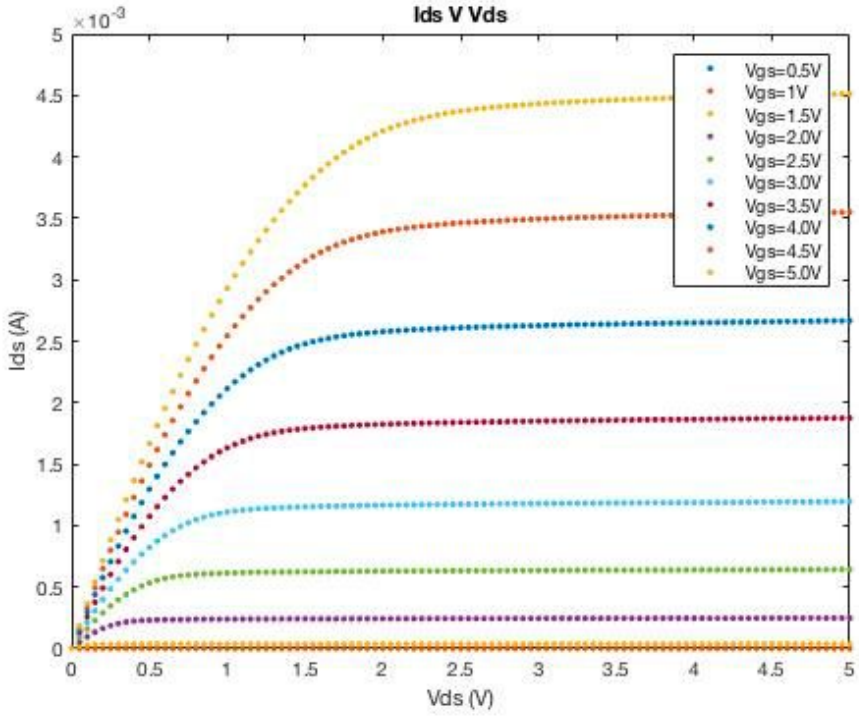
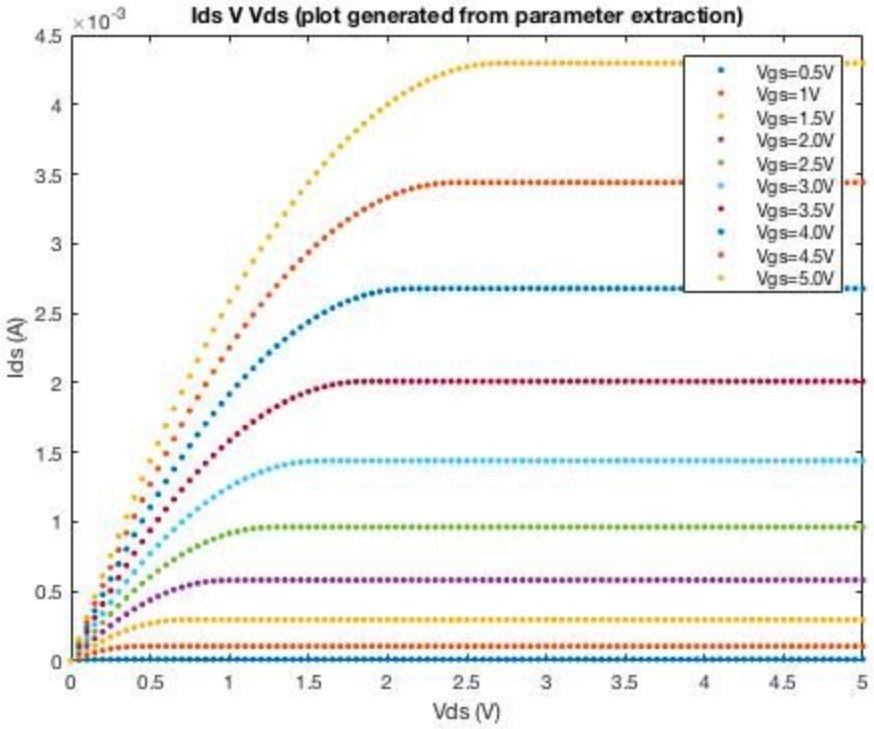


Figure 4. log(Ids) V. Vgs. The left plot is IDmeasured, the right plot is IDmodel.

When Vgs < Vth, IDmodel is insensitive to Vds. Both IDmeasured and IDmodel show this behavior. As shown in plot above, the curves of Vds = 0.5 and Vds = 4.5 overlap in the first region of Vgs.

The point of divergence (threshold voltage) on both graphs is at 1.5 V. Although the Vth result from our parameter extraction is 0.2V, so we were very confused on why the ID calculated from those parameters resulted a plot that has its threshold point at 1.5V.





Our IDmodel seems to match the IDmeasure very well basing on the plots shown above. Ids values are similar for each curve, and the threshold seems to be close as well.

Our asymptotic validation also shows that IDmodel resulted has expected behaviors with small relative errors.

```
ASYMPTOTIC VALIDATION RESULT
VGS < Vth: ID behaves as an exponential function of VGS with
||error|| = 0
VGS > Vth: ID behaves as an quadratic function to VGS with ||error||
= 0.346306
 VGS < Vth or VDS > VDsat : ID is insensitive to VDS with ||error|| =
0
```