**

**ECE 4960: Computational and Software Engineering**

**Spring 2018**

**Programming Assignment 3: Parameter Extraction from Least-Square Fitting**

**1. Goal**

1. Construct a modular program that can accomplish least-square fitting and parameter extraction with simple graphical visualization.
2. Familiarize with different iterative schemes in nonlinear optimization.
3. Set up automatic verification in various levels of unit testing, numerical black-box testing and asymptotic testing as a first step towards an “expert” system.

**2 Background preparation**

To describe our complex world, often a model is created with the independent variables (*x1, x2, … xn*) and meaningful parameters (*a1, a2, …, am*). A “scalar” model can then be generally expressed as:

*Smodel = S*(*x1, x2, … xn*; *a1, a2, …, am*) (1)

We can often make measurements of *S* with given values of the independent variables (*x1, x2, … xn*).

*Smeasured* = *S*(*x1, x2, … xn*) (2)

*S* can be a vector of rank *l* as well, but we will restrict here for *S* to be a scalar. The least-square parameter extraction to construct the model parameters (*a1, a2, …, am*) is an optimization problem to minimize the following function *V* that describe the least-square fitting:

for all *i*-th cases with the same (*x1, x2, … xn*) (3)

To map the parameter extraction problem to the nonlinear equation solver, Eq. (3) is equivalent to solve the set of *m* nonlinear equations by:

(4)



As *V* is always positive regardless of (*x1, x2, … xn*) and (*a1, a2, …, am*), it is straightforward to derive that the Hessian matrix is positive definite and we will always reach a minimum (least in “least square”). If *S* is a vector, Eq. (4) will be changed to Eq. (5) with most procedures staying the same.

(5)



where represents taking the norm in Eq. (3). You can see that we will have the exact number of *m* equations for the *m* parameters. As *V* is nonlinear, even when *S* is linear[[1]](#footnote-1), there may be more than one set of solutions (*a1, a2, …, am*), but we will not have under-determined or over-determined problems. The purpose of the model parameter extraction is often for estimation of *S* for a given (*x1, x2, … xn*) that we do not have measurements. For example, *x* is a function of time *t*, we can use *S* for making predictions or regression. Or in power network, we can ask what *x* makes *S* unstable (by definition, you cannot measure the unstable situation, or in general, we often cannot perform experiments on large-scale power network anyway).



For a top-down programming practice, instead of knowing the form of the objective functions to construct your programs from bottom up, you will write the parameter extraction with a generic function of *Smodel*. You will construct tests for the generic function described in Tasks for validation.

I have used a circuit element example here, but you do not need to have a physical understanding and can treat the model description as a black box. It really does not matter if you have never worked with detailed circuit models for analog and digital designs to perform the parameter extraction task. However, if you do, you will learn some inside knowledge for the parameter optimization scheme inside an electronic CAD program. The four-terminal MOSFET device operations can be described by the **EKV** (Enz-Krummenacher-Vittoz) model[[2]](#footnote-2) with the four terminals of S (source), D (drain), G (gate) and B (bulk) denoted in the subscript:

(6)



where *IF* described the forward current injection at the source junction and *IR* described the reverse current injection at the drain junction. The independent variables are *VGB, VSB* and *VDB*. For our measurement files, S is shorted to B, so you really have just two independent variables of *VGB* and *VDB*. *VT* is a constant of 26mV at the room temperature. There are three remaining parameters: the saturation current *IS*, the bulk factor *κ* and the threshold voltage *Vth*. The EKV model is built from adjoining two asymptotic regions of *VGS << Vth* (exponential) and *VGS >> Vth* (quadratic) with the approximations:

(8)



We will use these asymptotic knowledge to perform variations in the parameter extraction and the validation as well. To finish the approximation in different operational regions, we have (the three equations below are just for your curiosity, and have nothing to play in the assignment):

Subthreshold saturation: *VGS < Vth*, *VDS > 3VT*. This is how your transistor turns off.

(9)



Above-threshold linear: *VGS > Vth*, *VDS < VDsat*, where



(10)



Above-threshold saturation: *VGS > Vth*, *VDS > VDsat*. This is where the transistor has large analog gain.

where (11)



Use of asymptotic forms seems to have more intuition, but there will be discontinuity in magnitude or derivative in the expression that can cause not only numerical problems but wrong small signal analysis. However, asymptotic forms are useful in intuitive understanding and validation.

You will extract the three parameters *IS*, *κ* and *Vth* from *Smodel* = *ID(VGS, VDS*; *IS,* *κ*, *Vth*) and *Smeasured* from the file outputNMOS that is typically used to characterize a MOSFET. The measurement recorded *ID(VDS)* sweeps at different *VGS* biases. You will use the Newton’s method and secant method for the optimization problem. You should put in stop criteria of non-convergence and report the failed parameter search.

**3 Tasks**

1. Prepare a direct full-matrix solver that has reasonable pivoting checks (use the classnote ill-conditioned matrices for verification). This needs to solve matrices with rank up to 4.
2. Perform a validation of your parameter extraction program for *y = c0xm* (power law), where *c0* will be 10 and *m* will be −0.5. This is can be done most easily with *S* = log(*y*), *x1* = log(*x*), *a0* = *c0* and *a1 = m*. Generate 10 samples of *Smeasured* from internally calculated values with random noises within 10 – 20%.
3. Download the measurement data file from the blackboard. Notice that the first line is a header description. Use a graphics tool of your choice (Matlab, Excel Visual C++ or OpenGL) to plot *ID* vs. *VDS* with the different values of *VGS*.
4. Use *Smodel* = *ID(VGS, VDS*; *IS,* *κ*, *Vth*) in the EKV model and all available measured points as *Smeasured*. Use a quasi-Newton method to find the best fit of *IS,* *κ*, and *Vth*. You can use an initial guess of *IS* = 10-7A, *κ = 1*, and *Vth* = 1V. Repeat the same search with a secant method. Report the absolute deviation ||*V*||2 from Eq. (3). Implement an automated check of the quadratic convergence in||*V*||2 and the increment vector magnitude ||Δ||2, i.e., the absolute and the relative residual. Compute also the parameter sensitivity Δ*Sai* for *IS,* *κ*, and *Vth*.

(12)



1. You probably notice that when the current is very small, it will not affect the norm much. Repeat Task 3 by *Smodel* = *ID(VGS, VDS*; *IS,* *κ*, *Vth*)/*IDmeasured*.



1. Convergence to a solution is sensitive to the starting point. As we have only three parameters, full parameters search is still possible. Search the full gridded regions of *IS* in (10-8A, 3×10-8A, 10-7A, 3×10-7A, 10-6A, 3×10-6A, 10-5A, 3×10-5A), *κ* in (0.5, 0.6, 0.7, 0.8, 0.9) and *Vth* in (0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0). Which region gives the smallest ? Sometime this is used together with the Newton methods as candidates of initial guesses when the parameter space is not large, or the full parameter space is not searched, but just sampled by the Monte Carlo method. If convergence is not achieved in Task 4, use results of this search as a guide to the starting point selection.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *IS* | *κ* | *Vth* |  |  | *ΔSIS* | *ΔSκ* | *ΔSVth* |
| Task 4 (quasi Newton) |  |  |  |  |  |  |  |  |
| Task 4 (secant) |  |  |  |  |  |  |  |  |
| Task 5 (quasi Newton) |  |  |  |  |  |  |  |  |
| Task 5 (secant) |  |  |  |  |  |  |  |  |

1. Visualization often remains an important tool, not printout, as human perception of graphics can handle much more data than reading text. For Task 3 with the quasi-Newton method, generate the plots of log(*ID*) vs. *VGS* with two *VDS*, and *ID* vs. *VDS* with the 10 values of *VGS* for both *IDmeasured* and *IDmodel*. Sample plots are shown below. However, an expert system will minimize the user participation but translate human expertise to validation. Figure out how to execute the following validation checks that are often performed previously by factory engineers.

* For *VGS* < *Vth*, *IDmodel* should be an exponential function of *VGS* with *κ* < 1 and nearly insensitive to *VDS*.
* For *VGS* > *Vth* and *VDS* > , *IDmodel* should be quadratic to *VGS* and insensitive to *VDS*, or . This is the shape of the family curve in *ID(VDS)* with *VGS* as parameters.



* For *VGS* > *Vth* and *VDS* < *VDsat*, *IDmodel* should be quadratic to *VDS*.

**4 Execution and due dates**

* This project can be done by groups of 2 students.
* I expect the time of your programming to be comparable to Project 2.
* Export your .cpp, .h, and report to be clearly contained in your Git directory before 4/1 end of day. In addition to the required output results and plots, report should mention clearly the inputs and outputs that are useful in observing the behavior of the solver as well as testing code sections. This can include step sizes selected for numerical solvers, initial guesses, quadratic convergence observation, etc.
* The grading will be based on the following:
  + The design of solver, testing and validation, following the top down programming practice. (30%).
  + Coding and reporting the test results as well as comment on the observations. The tasks 2, 4 (automated convergence check), and 7 covers validation for solver, check for convergence, and correction of solution. Include checks for additional helper functions, if any. (30%)
  + Your understanding of the iterative nonlinear solver method and their convergence. Completing tasks 1, 3, 4, 5, 6, and reporting of the output solver results and plots, with comment on the observation. (30%)
  + The in-code documentation and readability of code. (10%)



**6 Further information for future exploration (Bonus, not required)**

If you are an analog circuit designer, you may be interested in the further physical model. Equation (6) is not complete for the Early effect that gives the output resistance, which is of critical importance in analog designs, but not in digital designs. For modern MOSFET, the output resistance is mostly determined by the drain-induced barrier lowering (DIBL) effect as a correction on the threshold voltage:

(13)



An additional parameter *λ* is introduced and can be further integrated into your parameter extraction method.

1. When *S* is linear to *x* and *a*, this can be the conventional least-square fit of a line. When *S* is nonlinear to *x* and *a*, *V* will be even more nonlinear to *x* and *a*. [↑](#footnote-ref-1)
2. The EKV model is Level 5 in SPICE, and is popular for analog and RF circuit designs as the model is C−∞ continuous, meaning that the functions and all of its derivatives are continuous in the operational range. Many other popular models such as BSIM use regional approximations and smoothing functions, which are unnecessarily complicated for our purposes here. [↑](#footnote-ref-2)