POLITECNICO DI MILANO

Project for the course “Advanced Programming for Scientific Computing”

Teacher: Luca Formaggia

2D MODELLING OF ORGANIC MIS CAPACITORS

Arianna Chiesa
Matr. 852319

June 27th, 2018

Contents

Chapter 1

Introduction

1.1 Introduction to the code

The present work started from an already existing code developed for the simulation of one-dimensional Organic Thin Film Transistors written in Octave.

Thus the first step for this work has been the conversion, in Octave, of the existing code from the 1D case to the 2D case. This has been accomplished by means of the Octave version of the *Bim* library, used to assemble diffusion, reaction and advection matrices needed for the simulation.

The second step in the development of the project has been the "translation" of the code from the Octave language to the C++ language. In this second step too, the writing of the code has been helped by the *Bim++* library (the C++ version of the Octave Bim library).

Note that for the Octave version of the 2D model, it has been used a triangular 2D mesh, while for the C++ version the mesh is quadrangular and it is built by means of the *p4est* library.

A detailed description on how to run the program is reported in the README file.

1.1.1 Choices of implementation

The general structure of the code, that is functions composing it and data structures, has been kept as similar as possible to that of the original 1D Octave code. This choice has been made so as to let future users to easily deal with both versions (the 1D-Octave version and the 2D-C++ version).

1.1.2 Data structures

The code is composed by a class named `Probl`. Its main application is the modelling and the simulation of the physical operation of organic electronic devices.

Indeed, through suitable methods, it stores the geometrical informations of the required device, the parameters of the material which the device is made of and other variables useful for the simulations.

Details about the class and its methods are provided in the following chapter.

1.2 Introduction to organic electronics

1.2.1 Organic semiconductors

The major difference between inorganic and organic materials lays in their molecular structure. Organic materials are carbon-based materials and their molecules are held together by weak van der Waals' interactions. This often leads to topological and energetic disorder in organic semiconductors. In inorganic materials instead, there are covalent bonds between molecules, which are stronger than van der Waals' interactions and originate a perfectly regular, crystalline solid. As a consequence, organic semiconductors have a lower charge carrier mobility, that is the ability of charged particles to move in response to an electric field, than the inorganic ones. In organic semiconductors indeed charge transport occurs via thermally activated hopping events between strongly localized energetic sites.

A hopping mechanism is a phonon-assisted and thermally activated quantum tunnelling effect from one site to another neighbouring site.

Although organic semiconductors seem not to have good perfomances, they allow easy and low-cost fabrication of large area circuits. Besides they take advantage of mechanical flexibility, high transparency and bio-compatibility [1]. For these reasons, it is interesting and important keeping on investigating them.

1.2.2 FETs: mechanism of operation

Field-Effect Transistors (FETs) are at the basis of nowadays electronic circuits and processors. A FET is a three-terminal (source, drain and gate) component where the current flows between source and drain depending on applied voltage at the gate terminal (Fig.1.1).

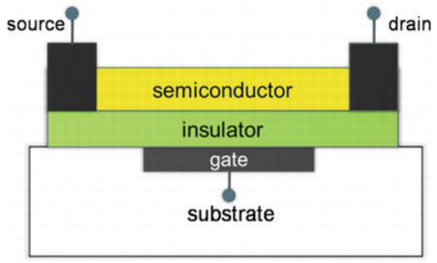


Figure 1.1: FET

The metallic gate, the insulator layer and the bulk semiconductor act as a capacitor, where the gate and the semiconductor form the plates among which the insulator lays acting as a dielectric. This is called a Metal-Insulator-Semiconductor (MIS) capacitor. Therefore when applying a bias across the plates, opposite and equal charges will accumulate at the insulator-semiconductor interface. When applying a higher bias than the threshold gate voltage V_T the device turns on. Then, applying a suitable drain-to-source voltage V_{DS} charge carriers motion from source to drain occurs [2].

Chapter 2

Class `Probl`

2.1 Constructor

As already mentioned in the Introduction (Sec.1.1.2), in class `Probl` are stored all the informations about the material, the geometry of the device, the structure of the mesh and the physical constants.

In order to set all of these parameters, the constructor makes use of the following methods: `Constants`, `Material`, `Device`, `Quad` and `Algor`.

2.1.1 `Constants` method

As the name of the method suggests, all the needed physical constants are set:

- Boltzmann's constant: $K_b = 1.38 \cdot 10^{-23}$;
- electron charge: $q = 1.602 \cdot 10^{-19}$ $[C]$;
- vacuum permittivity: $\varepsilon_0 = 8.85 \cdot 10^{-12}$ $[Fm^{-1}$;
- absolute temperature: T_0 $[K]$, which can be specified by the user, otherwise its default value is $T_0 = 300$ K;
- threshold voltage: $V_{th} = K_b \cdot \dfrac{T_0}{q}$ $[V]$.

2.1.2 `Material` method

This method stores the characteristic parameters of the material:

- low-field and low-charge density of electrons: $\mu_{0,n} = 4.29 \cdot 10^{-6}$ $[m^2V^{-1}s^{-1}]$;
- relative permittivity of the semiconductor: $\varepsilon_{sc,r} = 2.9$;

- relative permittivity of the insulator: $\varepsilon_{ins,r} = 2.83$;
- absolute permittivity of the semiconductor: $\varepsilon_{sc} = \varepsilon_0 \cdot \varepsilon_{sc,r}$ $[Fm^{-1}]$;
- absolute permittivity of the insulator: $\varepsilon_{ins} = \varepsilon_0 \cdot \varepsilon_{ins,r}$ $[Fm^{-1}]$;
- energy gap: $E_{gap} = 1.55$ $[eV]$;
- total number of available states (per unit volume): $N_0 = 10^{27}$ $[m^{-3}]$;
- metal to semiconductor barrier: Φ_B $[eV]$, whose default value is -0.54 if not specified by the user;
- degree of molecular disorder: $\sigma_n = 2.6$ $[J]$;
- $\sigma_{n,kT} = \dfrac{\sigma_n}{K_b \cdot T_0}$;
- intrinsic carrier concentration: n_i $[m^{-3}]$.

2.1.3 `Device` method

This method stores the geometrical informations of the device such as:

- length: $L = 1.4 \cdot 10^{-3}$ m;
- section: $s = 0.81 \cdot 10^{-6}$ m^2;
- width of the semiconductor: $t_{sc} = 3.5 \cdot 10^{-8}$ m;
- width of the insulator: $t_{ins} = 4.41 \cdot 10^{-7}$ m;
- values of the boundary edges of the device where the terminals (bulk and gate) lay: *contacts*$= \{2, 3\}$;
  in particular 2 and 3 are the indices of the sides of the quadrants where the bulk contact and the gate contact are (respectively).

The values reported for these parameters are those of default, but the user can set other values by specifying them in the `Probl` constructor.

One can also decide whether the insulator is present or not by means of the boolean variable *ins* and establish the rate of refinement of the mesh by setting the variable *maxcycle*, which determines the number of refinement cycles that the code must perform on the mesh.

Starting from this settings, the method generates the mesh through the *p4est* library. It creates the boolean vectors *insulator* and *scnodes* to keep a trace on elements and nodes of the mesh which are in the semiconductor or in the insulator (if present); while the vector *dnodes* stores the indices of the boundary nodes.

The number of trees constituting the mesh is also stored in the record $nTrees$.

Besides in the constructor of the class can be set:

- the drain potential V_D, in order to calculate the source-drain effective field $E_{field} = V_D/L \quad [V/m]$;
- the shift potential sensed at the gate terminal: $V_{shift} = 1.79738\ V$;
- a custom value of the potential applied at the gate contact: $V_G\ [V]$, whose default value is 0;
- a custom value of the potential applied at the bulk contact: $V_B\ [V]$, whose default value is 0.

2.1.4 `Quad` method

The `Quad` method computes nodes (gx) and weights (gw) for quadrature rules, which will be exploited to approximate integrals. The user can indicate the number of required nodes and weights by setting the parameter n in the constructor of class `Probl`, whose default value is equal to 101.

2.1.5 `Algor` method

Finally the `Algor` method sets:

- maximum allowed number of iterations for the Newton method (exploited to solve a Non-Linear Poisson problem), whose default value is $p_{max\_it} = 1000$;
- tolerance on the residual norm of the increment, used in the Newton algorithm as well, as another stopping criterion, whose default value is $p_{toll} = 1^{-10}$.

Other attributes of class `Probl`, which are set in the `NonLinearPoisson` method (presented in following) and not in the constructor, are:

- output potential: $V_{in}\ [V]$;
- output electron density: $n_{in}\ [m^{-3}]$;
- vector of the residual norm: *resnrm*;
- number of iterations: n_{iter}.

2.2 Other methods

Besides the five methods described above and exploited in the constructor of the class, other methods are implemented. Prior to describe them, it is worthwhile introducing the adopted notation and making some remark.

Adopted notation 1: Let be Ω the device domain. It is made of a semiconductor region Ω_{sc} and an insulator region Ω_{ins}, such that $\Omega = \Omega_{sc} \cup \Omega_{ins}$. In the code the domain is assumed to be rectangular with the following dimensions:

- the length of the device is equal to L, then the x coordinate varies in the range $[0, L]$;
- the width of the device is equal to t_{sc} for the semiconductor and t_{ins} for the insulator; therefore the y coordinate is assumed to vary in the range $[-t_{sc}, t_{ins}]$;
- the boundary conditions are enforced on the boundary nodes, which correspond to nodes whose ordinates are $-t_{sc}$ for the semiconductor boundary nodes and t_{ins} for those in the insulator. Therefore the following relations hold:
$$\begin{cases} \partial\Omega_{sc} = [0, L] \times \{-t_{sc}\} \ , \\ \partial\Omega_{ins} = [0, L] \times \{t_{ins}\} \ . \end{cases}$$

Adopted notation 2: Let $\varepsilon = \varepsilon_0 \varepsilon_r$, where ε_0 is the vacuum permittivity and ε_r the material relative permittivity. Hence the following expression holds:

$$\varepsilon = \begin{cases} \varepsilon_{sc} \ , & \text{in } \Omega_{sc} \\ \varepsilon_{ins} \ , & \text{in } \Omega_{ins} \end{cases}$$

A remark on boundary conditions:

- **Bulk contact: charge injection**
The bulk contact corresponds to coordinate $y = -t_{sc}$, where a Schottky barrier is present. A Schottky barrier is a potential energy barrier for electrons formed at a metal-semiconductor interface and it can be expressed as:
$$\Phi_B = -\frac{W_f - E_a}{q} \ ,$$
where W_f is the metal work function and E_a is the semiconductor electron affinity. Thus the boundary condition for the electric potential φ must taken into account a shift equal to Φ_B.

As already declared, Φ_B can be set by the user in the constructor of class `Probl`, if it is not specified, it is set to the default value $-0.54\ V$.

- **Gate contact: applied voltage**
  The gate contact corresponds to coordinate $y = t_{ins}$, where a shift of the electric potential V_{shift} is sensed due to permanents dipoles, fixed charges in dielectric or metal work function mismatch [3].

  V_{shift} can be set by the user as well and if it is not specified, it is set to the default value $1.79738\ V$.

Now that all the introductions have been made, the other methods can be presented.

2.2.1 `Laplace` method

Let φ be the electrostatic potential, then the Laplace problem can be introduced:

$$\begin{cases} -\nabla \cdot (\varepsilon \nabla \varphi) = 0\ , & \text{in } \Omega \\ \varphi = \Phi_B + V_B\ , & \text{in } \partial\Omega_{sc} \\ \varphi = V_{\text{shift}} + V_G\ , & \text{in } \partial\Omega_{ins} \end{cases}$$

where V_B and V_G can be set through the respective methods *set_VB* and *set_VG*.

The method performs the following steps:

- first the diffusion matrix, denoted by A, is assembled by means of the `bim2a_advection_diffusion` method of the Bim++ library;
- then the Dirichlet boundary conditions are enforced on the matrix A and the right-hand-side term f (which is null in this case) through the `bim2a_dirichlet_bc` method;
- at last the system is solved with the `mumps` solver and the output φ is stored in the record V_{in} of the class.

2.2.2 `LinearPoisson` method

Denoting with ρ the spatial charge density per unit volume $[C \cdot m^{-3}]$, it holds:

$$\rho = \begin{cases} -q\ (n - p + N_D)\ , & \text{in } \Omega_{sc} \\ 0\ , & \text{in } \Omega_{ins} \end{cases} \tag{2.1}$$

where n and p $[m^{-3}]$ are the charge carrier densities of electrons and holes respectively (per unit area), while N_D $[m^{-3}]$ is the net dopant concentration. The following hypothesis are assumed for the charge density [3]:

- the semiconductor is intrinsic, this means that the dopant concentration N_D is zero;
- the thermal generation of charge effects are negligible;
- the semiconductor is unipolar: the device operation is based mainly on the use of majority charge carriers; thus only n-type devices are considered ($p \approx 0$). Moreover also recombination/generation phenomena are neglected.

Hence the total charge density becomes:

$$\rho = \begin{cases} -qn \ , & \text{in } \Omega_{sc} \\ 0 \ , & \text{in } \Omega_{ins} \end{cases}$$

Finally the Linear Poisson problem is formulated as:

$$\begin{cases} -\nabla \cdot (\varepsilon \nabla \varphi) = \rho \ , & \text{in } \Omega \\ \varphi = \Phi_B + V_B \ , & \text{in } \partial\Omega_{sc} \\ \varphi = V_{\text{shift}} + V_G \ , & \text{in } \partial\Omega_{ins} \end{cases}$$

This method basically performs the same steps as the `Laplace` method, but in this case a reaction matrix is required: M. It is assembled exploiting the `bim2a_reaction` method of the Bim++ library. Then the boundary conditions are enforced on the matrix A and the right-hand-side term, which now is: $f = M \cdot \rho$. The system is finally solved with the `mumps` solver and the outputs φ and n are stored in the records V_{in} and n_{in}.

2.2.3 `NonLinearPoisson` method

The main difference between the Non-Linear Poisson and the Linear Poisson problem consists in the dependence of the electron density n with the potential.

In order to give details of this dependence, the following notation is introduced:

- φ_n is the electrochemical potential $[V]$, accounting for both electrical and chemical interactions between charges;
- $\phi = \varphi - \varphi_n$ is the chemical potential $[V]$;

- $g(\mathcal{E})$ represents the Density of States (DOS) function;
- $g(\mathcal{E})d\mathcal{E}$ represents the density of available quantum states (per unit volume) that may have energy within infinitesimal range $d\mathcal{E}$ of energies centered at $\mathcal{E}$;
- $f_D(\mathcal{E})$ denotes the occupation probability of the state having energy $\mathcal{E}$;
- $\mathcal{E}_F$ is the Fermi level which can be assumed $\mathcal{E}_F = -q\varphi_n$ without loss of generality.

The constitutive relation of the total amount of charge carriers per unit volume n is given by:

$$n = n(\varphi, \varphi_n) = \int_{-\infty}^{+\infty} g(\mathcal{E} + q\varphi) \cdot f_D(\mathcal{E} + q\varphi_n) d\mathcal{E} \ .$$

Thus n can be interpreted as the sum over all the admissible energies of the DOS function weighted on the probability of occupation of that state.

For organic materials the shape of the DOS function $g(\cdot)$ is assumed to belong to a family of given functions parametrized by a single parameter σ (DOS width) corresponding to the degree of disorder of the system. Thus the constitutive relation for n and its derivatives with respect to the chemical potential are:

$$n(\phi) = \frac{N_0}{\sqrt{\pi}} \int_{-\infty}^{+\infty} e^{-\eta^2} \left(1 + exp\left(\frac{\sqrt{2}\sigma\eta - q\phi}{k_B T}\right)\right)^{-1} d\eta \tag{2.2}$$

$$\frac{\partial n}{\partial \phi}(\phi) = -\frac{N_0 q}{\sigma} \sqrt{\frac{2}{\pi}} \int_{-\infty}^{+\infty} \eta e^{-\eta^2} \left(1 + exp\left(\frac{\sqrt{2}\sigma\eta - q\phi(\varphi)}{k_B T}\right)\right)^{-1} d\eta \tag{2.3}$$

Therefore in the Non-Linear Poisson problem ρ is no more defined as in Eq.(2.1) with n constant, but it is $\rho = -qn(\phi)$, with n given by the relation (2.2).

In this case, in order to compute ρ, another method of class `Probl` is exploited: the `org_gaussian_charge_n` method . In its turn, it makes use of the two methods `n_approx` and `dn_dV_approx` to compute relations (2.2) and (2.3) respectively.

Since these integral relations can be numerically approximated through Gaussian quadrature rules, nodes and weights computed through the `Quad` method (already described) are exploited.

The Non-Linear Poisson problem can be now introduced:

$$\begin{cases} -\nabla \cdot (\varepsilon \nabla \phi) = \rho(\phi) \ , & \text{in } \Omega \\ \phi = \Phi_B + V_B \ , & \text{in } \partial\Omega_{sc} \\ \phi = V_{\text{shift}} + V_G \ , & \text{in } \partial\Omega_{ins} \end{cases}$$

This method assembles the diffusion matrix A and the reaction matrix M with the same methods of the *Bim++* library used before. Since the problem is non-linear, it is linearized and solved thanks to the Newton method. The resulting linearized system is:

$$\begin{cases} \delta\phi = -J^{-1} \cdot \text{res} \ , & \text{in } \Omega \\ \delta\phi(-t_{sc}) = \Phi_B + V_B - \phi(-t_{sc}) \\ \delta\phi(t_{ins}) = V_{\text{shift}} + V_G - \phi(t_{ins}) \end{cases}$$

where

$$J = A - M \cdot d\rho$$

is the Jacobian matrix and

$$\text{res} = A\varphi - M \cdot \rho$$

is the residual vector.

Therefore a loop is performed until convergence on the residual norm of $\delta\phi$ is reached, or until the maximum allowed number of iterations p_{max_it} is reached. Note that the diffusion and the reaction matrices (A and M) do not change during the simulation, thus they are assembled only once before the starting of the loop. However, since the output potential ϕ changes at every iteration, the vector density of charge ρ and its derivative $d\rho$ must be computed at each iteration with the function `org_gaussian_charge_n` .

Then the residual vector and the Jacobian matrix can be assembled and the boundary conditions can be enforced on them.

After the solving of the system, through the `mumps` solver, the solution is updated: $\phi_{\text{out}} = \phi + \delta\phi$.

The computation of the residual norm of the increment is carried out by means of the method `Norm`.

At the end of the algorithm, all the outputs are stored in the records V_{in}, n_{in}, $resnrm$ and n_{iter} of the class.

Note that, in order to run the Newton algorithm, this method requires an initial guess V_{guess} for the potential, which is provided in input.

2.2.4 `CVcurve` method

This method computes the capacitance C of a MIS capacitor with varying applied voltages at the gate contact. For each value of V_G, a Non-Linear Poisson problem is solved through the method newly introduced.

Then the `CVcurve` method takes as input the output potential $\bar{\varphi}$ of the non-linear problem, assembles and solves the following system:

$$\begin{cases} -\nabla \cdot (\varepsilon \nabla \delta\varphi) - q\dfrac{\partial n}{\partial \varphi}(\bar{\varphi}) \cdot \delta\varphi = 0 \ , \quad \text{in } \Omega \\ \delta\varphi(-t_{sc}) = 0 \\ \delta\varphi(t_{ins}) = \delta V_G \end{cases}$$

The term $q\dfrac{\partial n}{\partial \varphi}(\bar{\varphi})$ is computed through the `org_gaussian_charge` method; the diffusion matrix A and the reaction matrix M are assembled by means of suitable methods of the *Bim++* library, as well as the enforcing of the Dirichlet boundary conditions.

At last the solution $\delta\varphi$ must be integrated on the contacts in order to compute the charge variation: $\delta Q = \left(-\varepsilon \dfrac{\partial \delta\varphi}{\partial z} \right)(t_{ins})$.

The result is exploited to computed the capacitance C with the relation:

$$C(V_G) = L \cdot \frac{\delta Q}{\delta V_G} \ ,$$

where δV_G can be assumed to be equal to 1 without loss of generality.

2.3 Save methods

Suitable methods for the visualization of the solutions are implemented:

- `savePoisson`: exports all the outputs of the Laplace, Linear Poisson and Non-Linear Poisson (V_{in}, n_{in}, $resnrm$ and n_{iter}) in a *.gz* file for the visualization of the results in Octave;

- `saveCV`: the vectors of all the applied gate voltages and the respective computed capacitances are exported in a file for the visualization of the results in Octave.

2.4 `set_T0` method

The only *set-method* provided in the class is the one related to the temperature T_0. Indeed if the user needs to set a different value for T_0, different from that furnished in the constructor of the class `Probl`, it has to use the `set_T0` method, since other parameters (depending on the temperature) must be changed as well. These parameters are the threshold voltage V_{th} and $\sigma_{n,kT}$.

In the project are also provided Octave scripts to be run for the visualization of the output data.

All the implementation has been carried out by consulting [4].

Chapter 3

Tests

3.1 Mesh

The project contains three main files named `test1_mesh`, `test2_mesh` and `test3_mesh`. Each of them are example tests for the building of the mesh.

Test 1 builds a mesh composed by a single row of 400 quadrants (as default) and no refinement is performed.

Starting from a mesh equal to the one in test 1, test 2 performs only a refinement cycle on it and the final mesh results to have two rows of 800 quadrants along the $y-$axis, with an amount of: 1600 total quadrants; 3 rows of nodes on the $x-$axis with coordinates 0, $L/2$ and L, each row constituted by 801 nodes on the $y-$axis.

Quadrants and nodes are divided into the two regions (semiconductor and insulator) as follows: 1437 nodes in the semiconductor and the remaining nodes in the insulator; 1434 quadrants in the semiconductor and the remaining quadrants in the insulator.

Finally test 3 builds a very refined mesh performing 5 cycles of refinement. Those tests have been implemented in order to verify that `Device` method in class `Probl` correctly builds the mesh.

In each tests the mesh is exported in a file *.vtu* and can be visualized with an external program, for example *Paraview*.

A remark on the mesh For all the numerical results presented in the following, the employed mesh, which discretizes the rectangular domain, is the one built in test 2.

3.2 Laplace problem

The differential problem presented in Sec.2.2.1, is solved in the file `test_Laplace` with different boundary conditions:

1. First, both at the bulk and at the gate terminal, a potential equal to $\Phi_B = -0.54\ V$ is applied:

   $$\varphi = -0.54\ , \quad \text{on } \ \partial\Omega = \partial\Omega_{sc} \cup \partial\Omega_{ins}$$

   It has been decided to enforce symmetric boundary conditions in order to verify that the method provides a correct outcome in a trivial situation. What it is expected is a constant potential distribution equal to $-0.54\ V$ in each node of the mesh.

   Indeed, as one can see in Fig.3.1(a), the soluition presents the expected behaviour. The same test has been also performed with Octave code (2D version) so as to verify that both codes provide the same solution. Comparing Fig.3.1(a) (C++ output) and Fig.3.1(b) (Octave output) the two solutions result to coincide.

2. Then asymmetric boundary conditions are imposed: Φ_B at the bulk and V_{shift} at the gate.

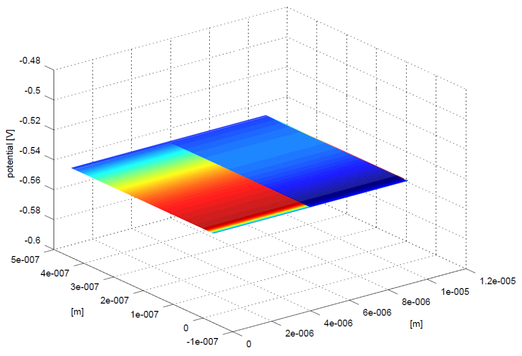
   $$\begin{cases} \varphi = -0.54\ , & \text{in } \partial\Omega_{sc} \\ \varphi = 1.79738\ , & \text{in } \partial\Omega_{ins} \end{cases}$$

   As one can notice, again the results obtained with the C++ code (Fig.3.1(c)) and the Octave code (Fig.3.1(d)) coincide.

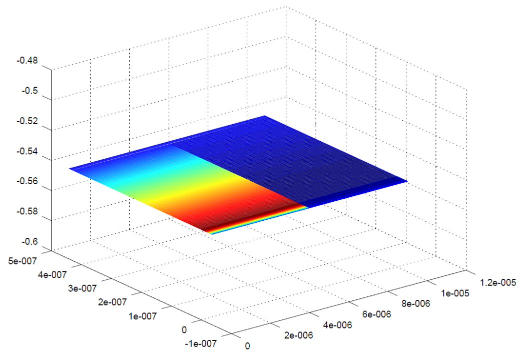
   In this case the potential has a linear growth, with different slopes in the semiconductor and in the insulator region. Evidence of such difference in the linear growth is depicted in Fig.3.2: the value of the potential is on the $y-$axis, while the nodes of the mesh are on the $x-$axis. Until the 1437^{th} mesh node, nodes are in the semiconductor region where a smoother growth is shown; from the 1437^{th} to the 2403^{rd} node, nodes belong to the insulator region, where the slope is steeper.

3.3 Linear Poisson problem

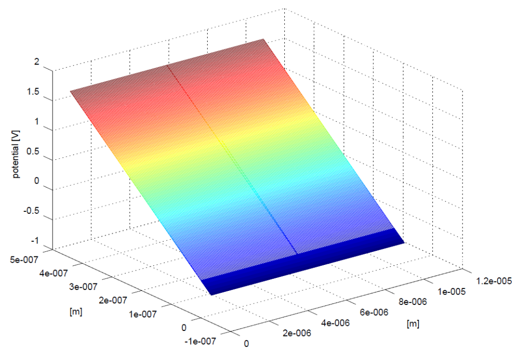
Tests on the Linear Poisson problem, presented in Sec.2.2.2, are performed by means of the main file `test_LinPoisson`. Again symmetric boundary conditions are enforced at first and then asymmetric boundary conditions are imposed.



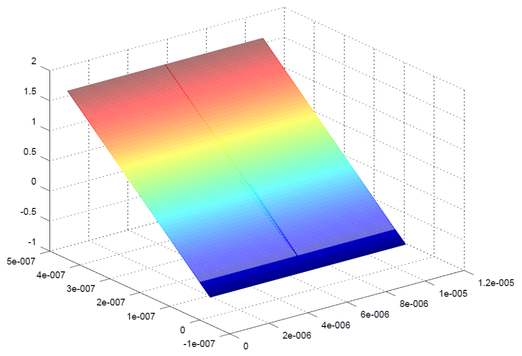
(a) C++ output for the potential with asymmetric Dirichlet boundary conditions.



(b) Octave output for the potential with symmetric Dirichlet boundary conditions.



(c) C++ output for the potential with asymmetric Dirichlet boundary conditions.



(d) Octave output for the potential with asymmetric Dirichlet boundary conditions.

Figure 3.1: Laplace problem: comparison between outputs of the C++ code and the Octave code.

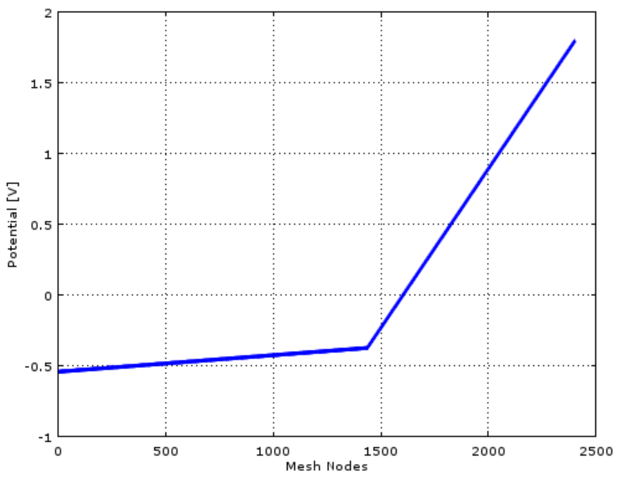


Figure 3.2: One-dimensional plot of the solution of the Laplace problem.

1. A potential equal to $\Phi_B = -0.54\ V$ is applied both at the bulk and at the gate at first:

$$\varphi = -0.54\ , \quad \text{on} \ \ \partial\Omega = \partial\Omega_{sc} \cup \partial\Omega_{ins}$$

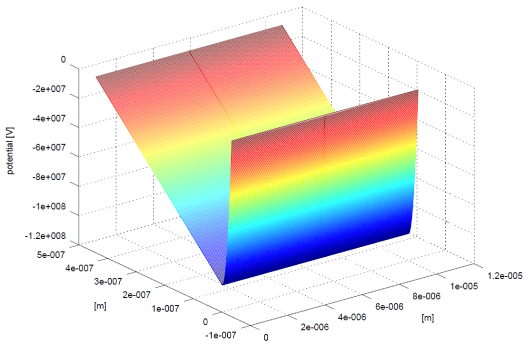
   In this case the solution is not perfectly constant, but it shows a minimum at the semiconductor-insulator interface. Making reference to Figg.3.3(a) and 3.3(b), one can see that both the solutions, provided by the C++ code and the Octave code, coincide.

2. Then asymmetric boundary conditions are imposed: Φ_B at the bulk and V_{shift} at the gate.

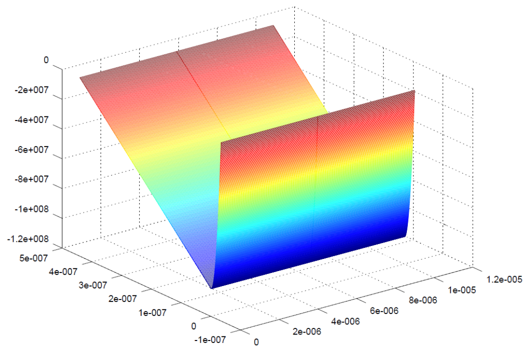
$$\begin{cases} \varphi = -0.54\ , & \text{in } \partial\Omega_{sc} \\ \varphi = 1.79738\ , & \text{in } \partial\Omega_{ins} \end{cases}$$

   Exactly like in the symmetric boundary conditions case, the outputs show a minimum at the semiconductor-insulator interface. See Fig.3.3(c) for the C++ output and Fig.3.3(d) for the Octave output.

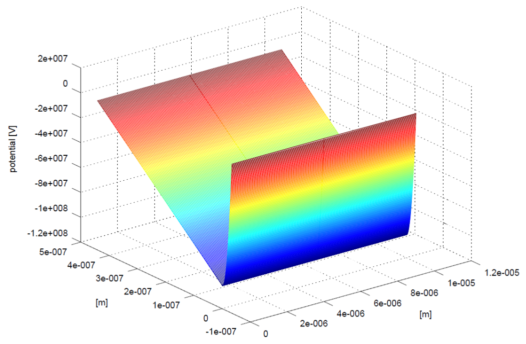
It is worthwhile noticing that, in both Linear Poisson tests (the one with symmetric boundary conditions and the one with asymmetric boundary conditions), the trend of the potential in the two regions is different. In the semiconductor region it has a parabolic growth, while in the insulator region it shows a linear growth, as depicted in Fig.3.4. The parabolic trend appearing in the semiconductor region is due to the present of charges, which are absent in the insulator.



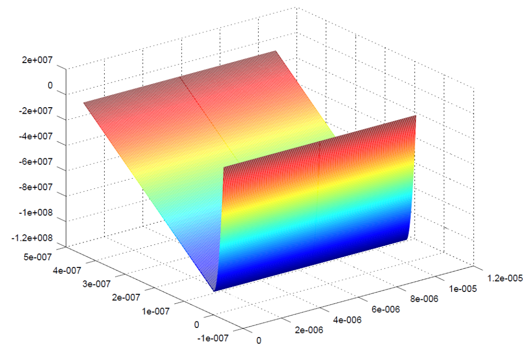
(a) C++ output for the potential with asymmetric Dirichlet boundary conditions.



(b) Octave output for the potential with symmetric Dirichlet boundary conditions.



(c) C++ output for the potential with asymmetric Dirichlet boundary conditions.



(d) Octave output for the potential with asymmetric Dirichlet boundary conditions.

Figure 3.3: Linear Poisson problem: comparison between outputs of the C++ code and the Octave code.

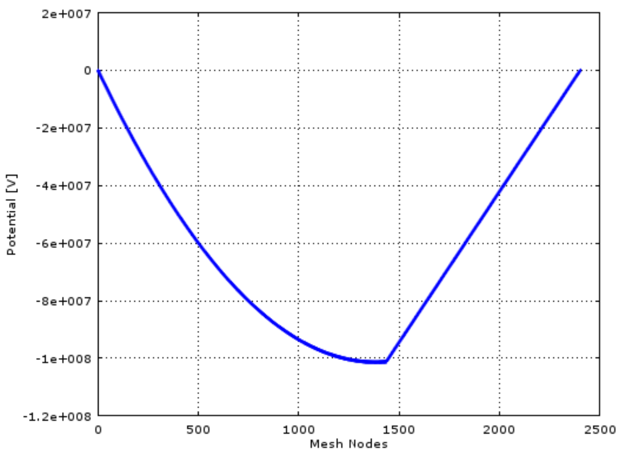


Figure 3.4: One-dimensional plot of the solution of the Linear Poisson problem.

3.4 Non-Linear Poisson problem

File `test_NLPoisson` is used to run tests on the Non-Linear Poisson problem (Sec.2.2.3).

It is worthwhile noticing that an initial guess on the potential must be provided to the `NonLinearPoisson` method in order to perform the loop for the Newton algorithm which computes the final solution. This initial guess has been taken constant and equal to $\Phi_B = -0.54\ V$ and a representation of it is provided in Fig.3.5.

As done with the previous test cases, symmetric boundary conditions are enforced at first and then asymmetric boundary conditions are imposed.

1. Symmetric boundary conditions impose Φ_B both at the bulk and at the gate contact:

$$\varphi = -0.54\ , \quad \text{on} \ \ \partial\Omega = \partial\Omega_{sc} \cup \partial\Omega_{ins}$$

   In this case the provided initial guess already satisfies the boundary conditions, thus a constant solution is expected. Indeed the output is almost flat with a minimum at the semiconductor-insulator interface equal to $-0.54007\ V$, as shown in Fig.3.6(a). Note that the output obtained with the Octave code is not provided, nonetheless its solution resulted to coincide with the one provided by the C++ code.

2. Then asymmetric boundary conditions are imposed: this time, contrary to the previous test cases, a potential equal to $V_G = 35\ V$ is applied

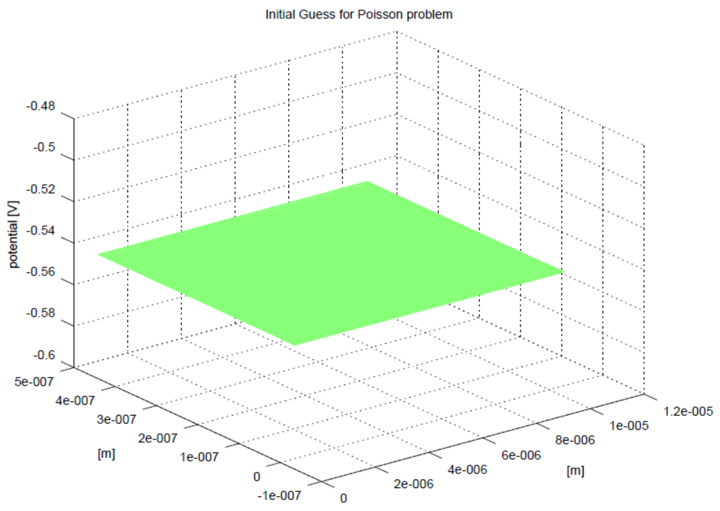


Figure 3.5: Representation of the initial guess of the Non-Linear Poisson problem.

together with the shift in the potential V_{shift} at the gate terminal. Thus the final boundary Dirichlet conditions enforced are:

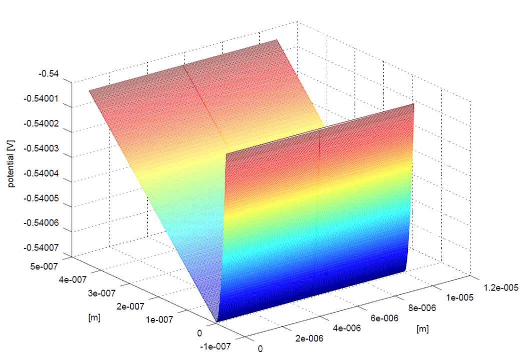
$$\begin{cases} \varphi = -0.54 \ , & \text{in } \partial\Omega_{sc} \\ \varphi = 36.79738 \ , & \text{in } \partial\Omega_{ins} \end{cases}$$

In Fig.3.6(b) it can be easily noticed that the potential has a different slope in the semiconductor region and in the insulator region. Indeed the slope in the insulator is steeper than in the semiconductor as better highlighted by Fig.3.6(c).

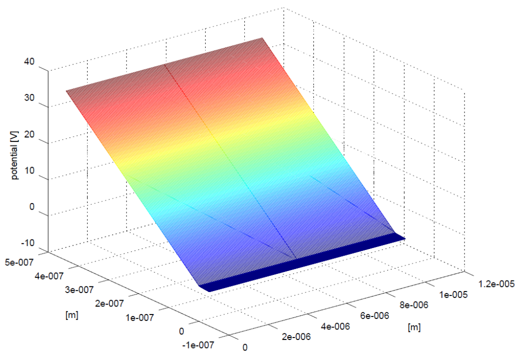
The Newton iterative method used to solve the system results to converge in 32 iterations. The exploited stopping criterion is that of the residual norm of the increment δ and its trend is reported in Fig.3.6(d).

Considering that the electron charge density n depends on the potential φ, plots of the final computed electron charge density in the device are provided. In Fig.3.6(e) is depicted the 3D final output of the electron charge density, while Fig.3.6(f) shows its 1D plot to better understand its trend.

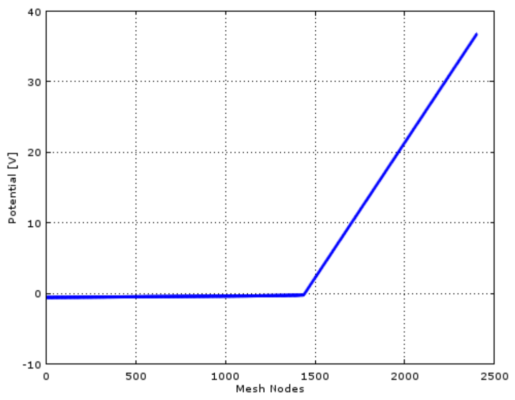
Note that: in the 3D plots the insulator is on the left, while in the 1D plots it is on the right.



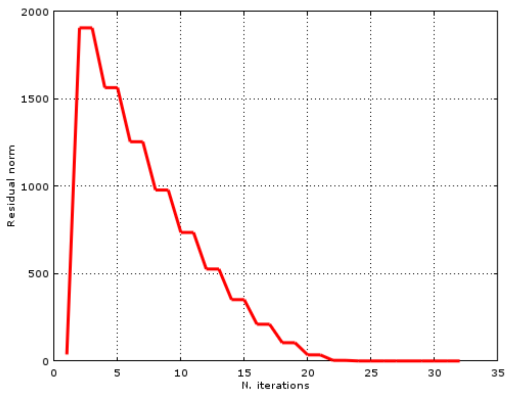
(a) Output for the potential with symmetric Dirichlet boundary conditions.



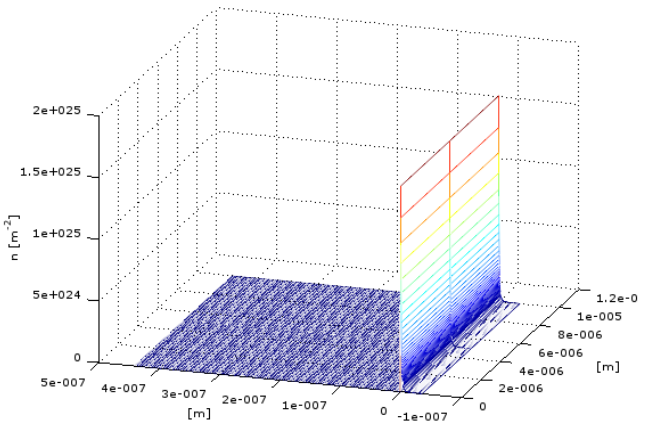
(b) Output for the potential with asymmetric Dirichlet boundary conditions.



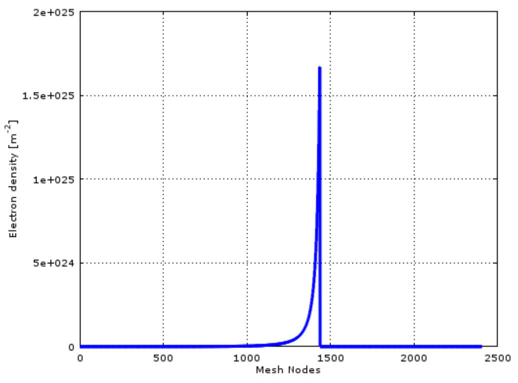
(c) One-dimensional plot of the solution of the Non-Linear Poisson problem.



(d) Plot of the residual norm.



(e) Output for the electron charge density with asymmetric Dirichlet boundary conditions.



(f) One-dimensional plot of the electron charge density.

Figure 3.6: Output of the Non-Linear Poisson problem.

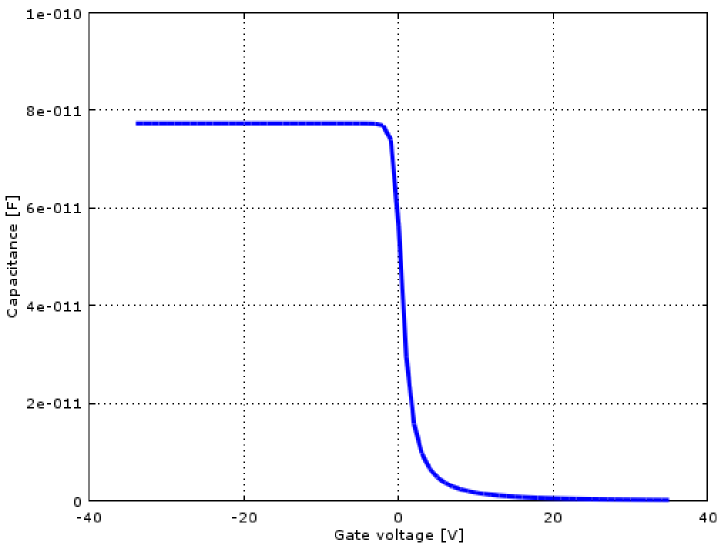


Figure 3.7: Plot of the MIS capacitance at different values of the gate potential.

3.5 CV curve

The last test performs is implemented in the main `test_CVcurve`. With different values of V_G, ranging from $-35\ V$ to $35\ V$, the capacitance of the MIS capacitor is computed.

As expected, when the applied voltage at the gate terminal is negative, a strong potential difference across the capacitor is sensed. This potential difference leads to an accumulation of charges at the semiconductor-insulator interface. With the increasing of the gate voltage, the potential difference becomes smaller and the charges begin to leave the interface with a consequent decreasing of the capacitance.

Note that for positive gate voltages the capacitance is not zero, it is still positive although it cannot be perfectly seen by the plot (see Fig.3.7).

3.6 Conclusions

In all the three test cases presented so far (Laplace problem, Linear Poisson problem, Non-Linear Poisson problem), an agreement between the results obtained with the C++ code and the Octave code has been confirmed. Therefore the `Probl` class is verified to work properly.

Bibliography

[1] URL: `http://wwwmayr.in.tum.de/konferenzen/Jass08/courses/4/buth/slides_buth.pdf`.

[2] Pasquale Claudio Africa. “Numerical modeling of organic thin film transistors”. Politecnico di Milano, 2015. URL: `http://hdl.handle.net/10589/112524`.

[3] Francesco Maddalena et al. “Assessing the width of Gaussian density of states in organic semiconductors”. In: *Organic Electronics* 17 (2015), pp. 304 –318. ISSN: 1566-1199.

[4] Stanley B. Lippman, Jose Lajoie, and Barbara E. Moo. *C++11 Primer*. Ed. by Addison-Wesley Professional ©. 5th. 2014.