

Estrazione della Densità degli Stati in semiconduttori organici

Pasquale Africa

19 novembre 2014

Introduzione

Motivazione

Sommario

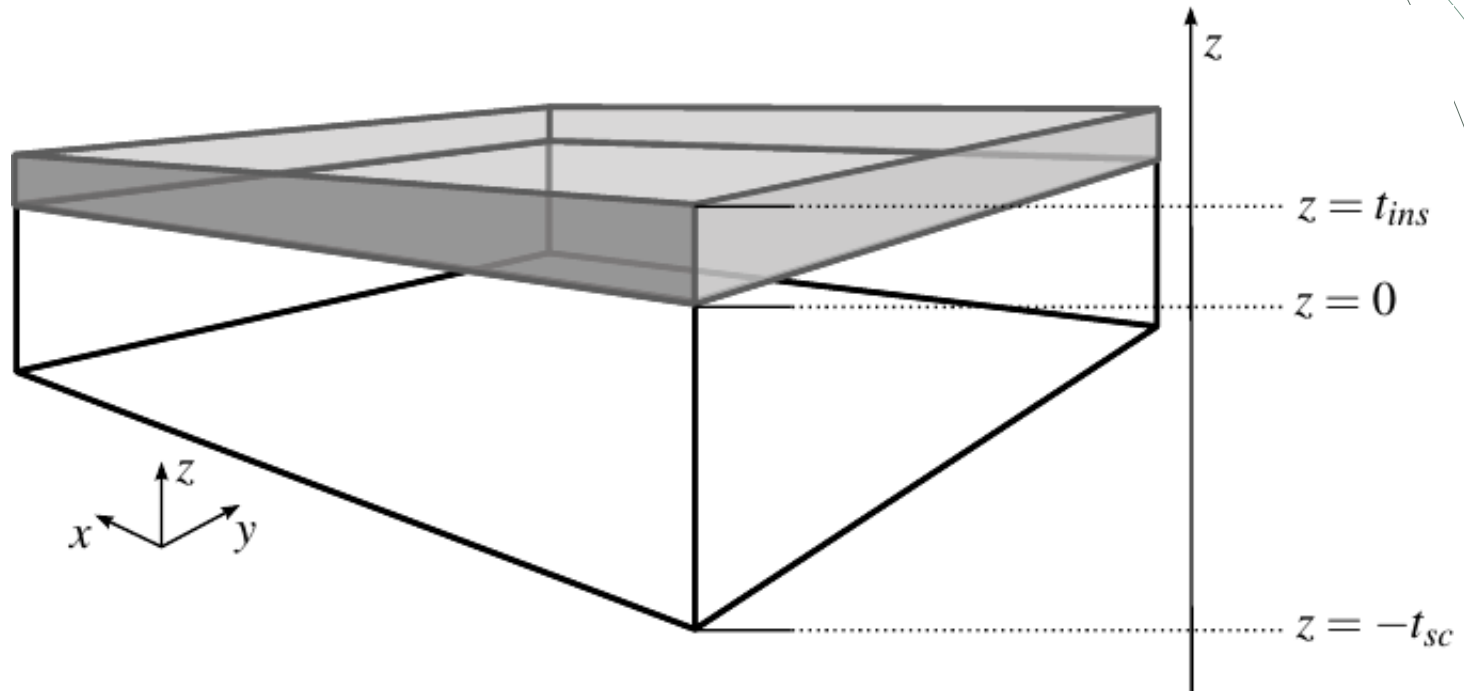
Modello

Metodi numerici

Implementazione

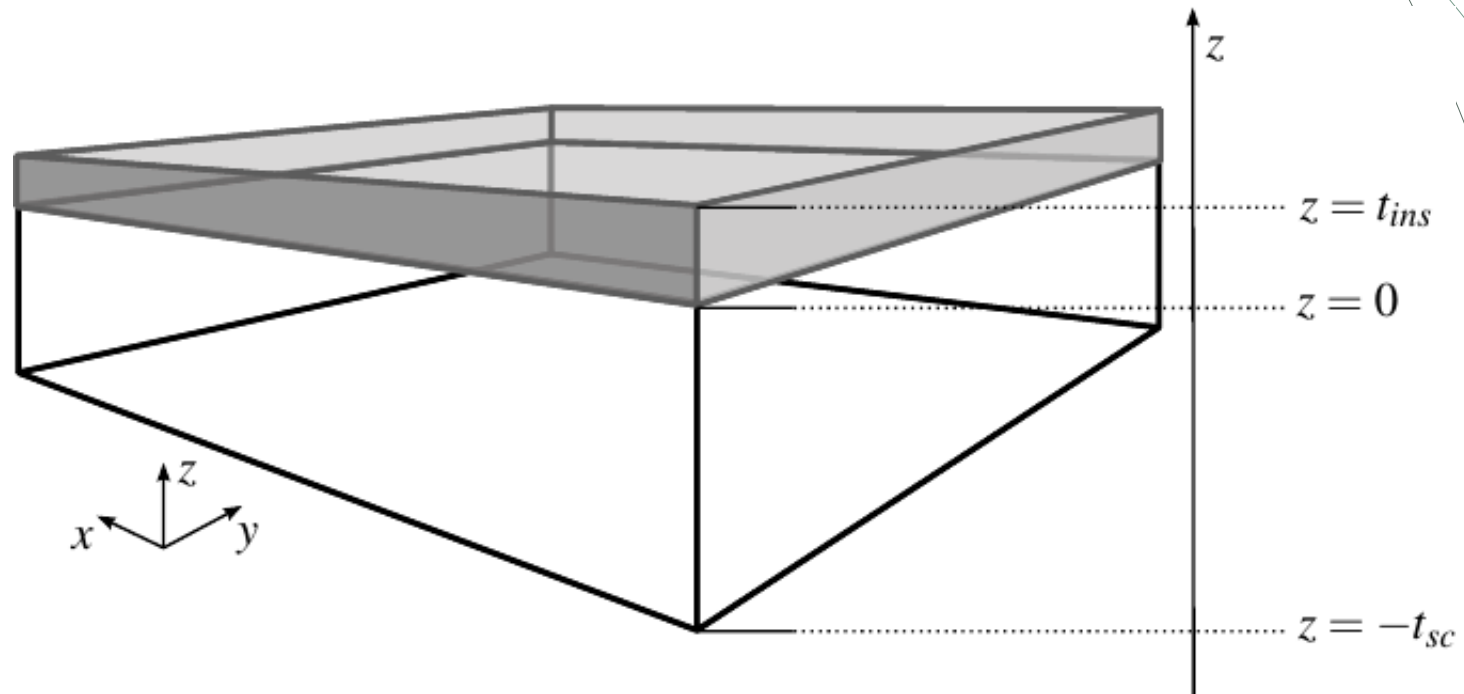
Risultati

Condensatore **MIS** (**M**etal - **I**nsulator - **S**emiconductor)



Introduzione
Motivazione
Sommario
Modello
Metodi numerici
Implementazione
Risultati

Condensatore **MIS** (**M**etal - **I**nsulator - **S**emiconductor)



Semiconduttore a base organica:

- basso costo
- possibilità di ottenere dispositivi flessibili (display, smart-card etc.)
- prestazioni paragonabili

Introduzione

Motivazione

Sommario

Modello

Metodi numerici

Implementazione

Risultati

Sistema drift-diffusion:

$$\begin{cases} -\nabla \cdot (\epsilon \nabla \varphi) = \rho & \text{in } \Omega \\ q \frac{\partial n}{\partial t} - q \nabla \cdot (D_n \nabla n - n \mu_n \nabla \varphi) + q R = q G & \text{in } \Omega_{semic} \\ q \frac{\partial p}{\partial t} - q \nabla \cdot (D_p \nabla p + p \mu_p \nabla \varphi) + q R = q G & \text{in } \Omega_{semic} \end{cases}$$

dove:

- φ è potenziale elettrico [V];
- n è la concentrazione volumetrica di elettroni [m^{-3}];
- p è la concentrazione volumetrica di lacune [m^{-3}].

Introduzione

Motivazione

Sommario

Modello

Metodi numerici

Implementazione

Risultati

Sistema drift-diffusion:

$$\begin{cases} -\nabla \cdot (\epsilon \nabla \varphi) = \rho & \text{in } \Omega \\ q \frac{\partial n}{\partial t} - q \nabla \cdot (D_n \nabla n - n \mu_n \nabla \varphi) + q R = q G & \text{in } \Omega_{semic} \\ q \frac{\partial p}{\partial t} - q \nabla \cdot (D_p \nabla p + p \mu_p \nabla \varphi) + q R = q G & \text{in } \Omega_{semic} \end{cases}$$

dove:

- φ è potenziale elettrico [V];
- n è la concentrazione volumetrica di elettroni [m^{-3}];
- p è la concentrazione volumetrica di lacune [m^{-3}].

Extended Gaussian Disorder Model:

$$\mu_n(T, \nabla \varphi, n) = \mu_0(T) \cdot g_1(\nabla \varphi) \cdot g_2(n), \quad \boxed{g_1, g_2 \propto \text{DOS}}$$

$$D_n = g_3 V_{th} \mu_n$$

Introduzione

Motivazione

Sommario

Modello

Metodi numerici

Implementazione

Risultati

- Problema di **identificazione dei parametri**
 - disordine molecolare σ , λ
- estensione di un codice Octave

Introduzione

Motivazione

Sommario

Modello

Metodi numerici

Implementazione

Risultati

- Problema di **identificazione dei parametri**
 - disordine molecolare σ , λ
- estensione di un codice Octave
- equazione di Poisson integro-differenziale non lineare
 - relazioni costitutive per la Density of States

Introduzione

Motivazione

Sommario

Modello

Metodi numerici

Implementazione

Risultati

- Problema di **identificazione dei parametri**
 - disordine molecolare σ , λ
 - estensione di un codice Octave
 - equazione di Poisson integro-differenziale non lineare
 - relazioni costitutive per la Density of States
-
1. Modello matematico
 2. Metodi numerici: linearizzazione, discretizzazione etc.
 3. Implementazione della libreria
 4. Risultati e conclusioni

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

Dalle equazioni di Maxwell:

$$\nabla \cdot \vec{D} = \rho$$

$$\nabla \times \vec{E} + \frac{\partial \vec{B}}{\partial t} = \vec{0}$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{H} - \frac{\partial \vec{D}}{\partial t} = \vec{J}$$

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

Dalle equazioni di Maxwell:

$$\begin{aligned}\nabla \cdot \vec{D} &= \rho \\ \nabla \times \vec{E} + \frac{\partial \vec{B}}{\partial t} &= \vec{0} \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{H} - \frac{\partial \vec{D}}{\partial t} &= \vec{J}\end{aligned} \quad \Rightarrow \quad \boxed{-\nabla \cdot (\epsilon \nabla \varphi) = \rho}$$

dove:

- φ è potenziale elettrico $[V]$;
- ϵ è la permittività elettrica del mezzo $[C \cdot V^{-1} \cdot m^{-1}]$;
- ρ è la densità di carica $[C \cdot m^{-3}]$;

Equazione di Poisson

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

$$\epsilon = \begin{cases} \epsilon_{semic} & \text{nel semiconduttore} \\ \epsilon_{ins} & \text{nell'isolante} \end{cases}$$

→ l'equazione vale in senso debole

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

$$\epsilon = \begin{cases} \epsilon_{semic} & \text{nel semiconduttore} \\ \epsilon_{ins} & \text{nell'isolante} \end{cases}$$

→ l'equazione vale in senso debole

$$\rho = \begin{cases} -q(n - p + N_A - N_D) & \text{nel semiconduttore} \\ 0 & \text{nell'isolante.} \end{cases}$$

dove:

- n, p concentrazioni di elettroni e lacune $[m^{-3}]$;
- N_A, N_D drogaggio (assunto nullo nel caso organico) $[m^{-3}]$.

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

$$f_D(\mathcal{E}) = \frac{1}{1 + \exp\left(\frac{\mathcal{E} - \mathcal{E}_F}{k_B \cdot T}\right)}$$

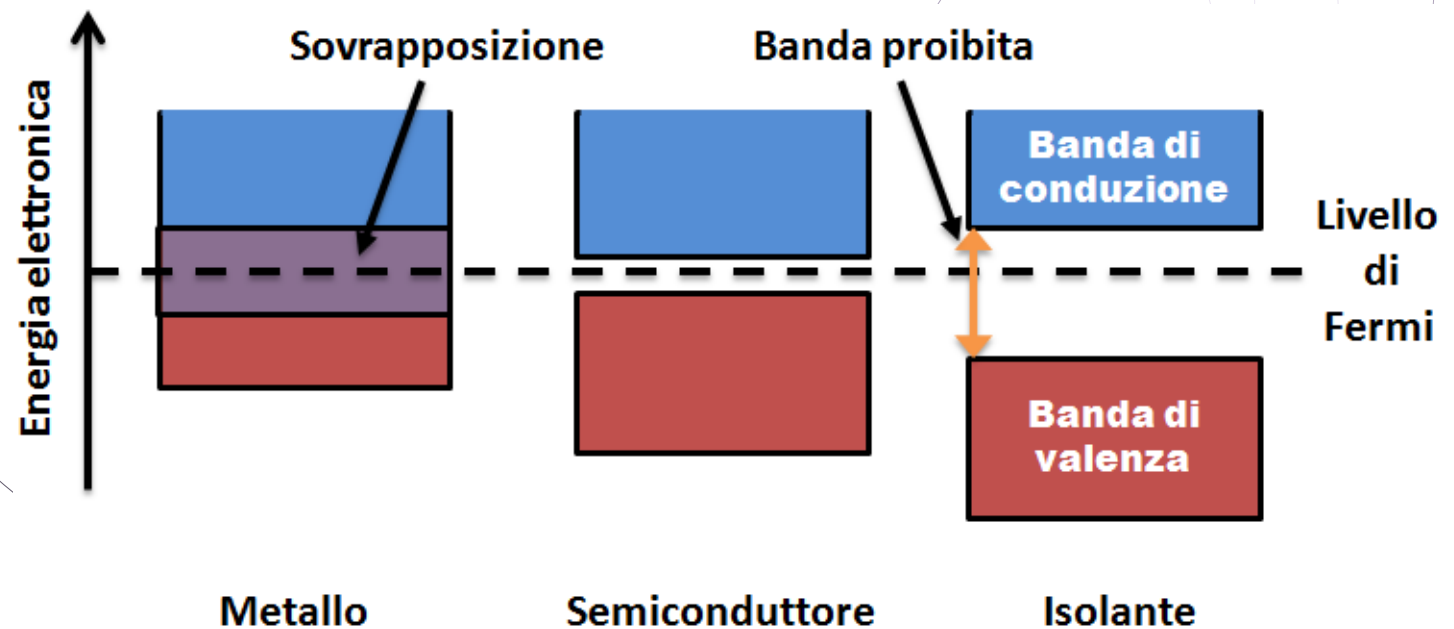
$f_D \approx$ numero di elettroni aventi energia \mathcal{E} (Pauli)

■ k_B costante di Boltzmann [$J \cdot K^{-1}$], T temperatura [K];

$$f_D(\mathcal{E}) = \frac{1}{1 + \exp\left(\frac{\mathcal{E} - \mathcal{E}_F}{k_B \cdot T}\right)}$$

$f_D \approx$ numero di elettroni aventi energia \mathcal{E} (Pauli)

- k_B costante di Boltzmann [$J \cdot K^{-1}$], T temperatura [K];
- \mathcal{E}_F livello di Fermi [J]: livello occupato di maggior energia in un sistema di fermioni alla temperatura di $0K$.



Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

Valgono:

$$n = \int_{-\infty}^{+\infty} g(\mathcal{E} - \mathcal{E}_{LUMO}) \cdot f_D(\mathcal{E} - \mathcal{E}_F) d\mathcal{E}$$

$$p = \int_{-\infty}^{+\infty} g(\mathcal{E}_{HOMO} - \mathcal{E}) \cdot [1 - f_D(\mathcal{E} - \mathcal{E}_F)] d\mathcal{E}$$

dove l'integrale è esteso ai livelli di energia ammissibili e:

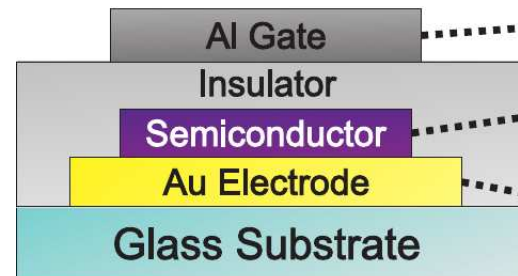
- $g \equiv$ DOS;
- \mathcal{E}_{HOMO} energia Highest Occupied Molecular Orbital;
- \mathcal{E}_{LUMO} energia Lowest Unoccupied Molecular Orbital:

$$\varphi = -\frac{\mathcal{E}_{LUMO}}{q} \quad \text{in } \Omega_{semic}.$$

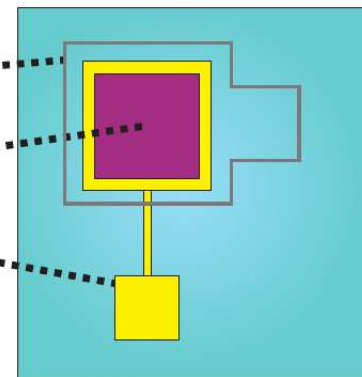
■ Approssimazione 1D:

$$x \gg z$$

Side View



Top View



■ effetti termici trascurabili (solo disordine)

■ regime quasi-statico:

- equilibrio ($\Rightarrow \mathcal{E}_F$ costante, assunto 0)
- no trasporto

■ semiconduttore intrinseco:

$$\rho = -q(n - p)$$

■ solo una specie di portatori (elettroni):

$$\rho = -qn$$

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

■ Gaussiana (singola o multipla)

■ Esponenziale

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

■ Gaussiana (singola o multipla) :

$$g_{\sigma}(\cdot) = \frac{N_0}{\sqrt{2\pi}\sigma} e^{-\frac{(\cdot)^2}{2\sigma^2}}$$

■ Esponenziale :

$$g_{\lambda}(\cdot) = \frac{N_0}{\lambda} \exp\left(-\frac{(\cdot)}{\lambda}\right)$$

■ Gaussiana (singola o multipla) :

$$g_{\sigma}(\cdot) = \frac{N_0}{\sqrt{2\pi}\sigma} e^{-\frac{(\cdot)^2}{2\sigma^2}}$$

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

■ Esponenziale :

$$g_{\lambda}(\cdot) = \frac{N_0}{\lambda} \exp \left(-\frac{(\cdot)}{\lambda} \right)$$

$$n(\varphi) = \frac{N_0}{\lambda} \int_0^{+\infty} e^{-\alpha} \left(1 + \exp \left(\frac{\lambda\alpha - q\varphi}{k_B \cdot T} \right) \right)^{-1} d\alpha$$

Introduzione

Modello

Equazione di Poisson

Statistica di
Fermi-Dirac

Ipotesi

Relazioni costitutive

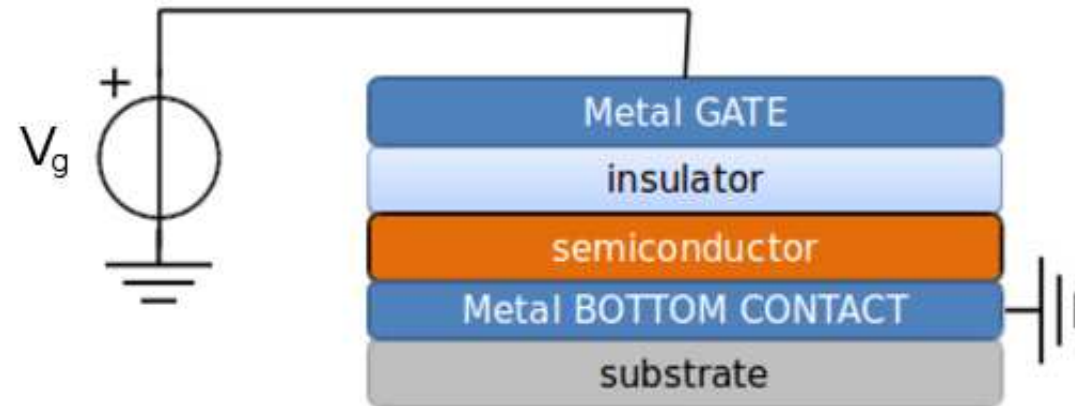
Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati



Il potenziale φ è fissato ai contatti elettrici \longrightarrow condizioni di Dirichlet.

■ Gate (tensione esterna):

$$\varphi = V_g + V_{shift} \quad \text{su } \Gamma_{ins}$$

■ Back (barriera di **Schottky**):

$$\varphi = \phi_F - \phi_B = -\phi_B \quad \text{su } \Gamma_{semic}$$

Introduzione

Modello

Equazione di Poisson

Statistica di

Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

$$\Omega_{semic} = (t_{semic}, 0]$$

$$\Omega_{ins} = (0, t_{ins})$$

$$\begin{cases} -(\epsilon\varphi')'(z) = -\frac{N_0 q}{\sqrt{\pi}} \int_{-\infty}^{+\infty} \frac{e^{-\alpha^2}}{\left(1 + \exp\left(\frac{\sqrt{2}\sigma\alpha - q\varphi(z)}{k_B \cdot T}\right)\right)} d\alpha & z \in \Omega_{semic} \\ -(\epsilon\varphi')'(z) = 0 & z \in \Omega_{ins} \\ \varphi(-t_{semic}) = -\phi_B \\ \varphi(t_{ins}) = V_g + V_{shift} . \end{cases}$$

→ equazione di Poisson **integro-differenziale non lineare**.

Introduzione

Modello

Equazione di Poisson

Statistica di

Fermi-Dirac

Ipotesi

Relazioni costitutive

Condizioni al contorno

Riepilogo

Metodi numerici

Implementazione

Risultati

$$\Omega_{semic} = (t_{semic}, 0]$$

$$\Omega_{ins} = (0, t_{ins})$$

$$\begin{cases} -(\epsilon\varphi')'(z) = -\frac{N_0 q}{\sqrt{\pi}} \int_{-\infty}^{+\infty} \frac{e^{-\alpha^2}}{\left(1 + \exp\left(\frac{\sqrt{2}\sigma\alpha - q\varphi(z)}{k_B \cdot T}\right)\right)} d\alpha & z \in \Omega_{semic} \\ -(\epsilon\varphi')'(z) = 0 & z \in \Omega_{ins} \\ \varphi(-t_{semic}) = -\phi_B \\ \varphi(t_{ins}) = V_g + V_{shift} . \end{cases}$$

→ equazione di Poisson **integro-differenziale non lineare**.

Dovrà essere:

1. linearizzata;
2. discretizzata.

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

L'equazione si può scrivere:

$$\mathcal{F}(\varphi) = 0$$

dove \mathcal{F} è un funzionale integro-differenziale non lineare.

→ **Metodo di Newton**

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

L'equazione si può scrivere:

$$\mathcal{F}(\varphi) = 0$$

dove \mathcal{F} è un funzionale integro-differenziale non lineare.

—→ **Metodo di Newton** :

occorre risolvere

$$\begin{cases} \mathcal{DF}(\varphi^{(k)}) [\delta\varphi^{(k)}] = -\mathcal{F}(\varphi^{(k)}) \\ \varphi^{(k+1)} = \varphi^{(k)} + \delta\varphi^{(k)} \end{cases}$$

fino a convergenza.

$\mathcal{DF}(\cdot)$ è la derivata secondo **Gâteaux** di \mathcal{F} .

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

$$\mathcal{DF}(\varphi)[\chi] = \lim_{\kappa \rightarrow 0} \frac{\mathcal{F}(\varphi + \kappa\chi) - \mathcal{F}(\varphi)}{\kappa} =$$

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

$$\begin{aligned}\mathcal{DF}(\varphi)[\chi] &= \lim_{\kappa \rightarrow 0} \frac{\mathcal{F}(\varphi + \kappa\chi) - \mathcal{F}(\varphi)}{\kappa} = \\ &= \lim_{\kappa \rightarrow 0} \frac{1}{\kappa} \left[-(\cancel{\epsilon\phi} + \kappa\epsilon\chi')' + \cancel{(\epsilon\phi)'} \right] + \\ &+ \lim_{\kappa \rightarrow 0} \frac{1}{\kappa} \frac{N_0 q}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} \left[\exp\left(-\frac{(\mathcal{E} + q(\varphi + \kappa\chi))^2}{2\sigma^2}\right) + \right. \\ &\quad \left. - \exp\left(-\frac{(\mathcal{E} + q\varphi)^2}{2\sigma^2}\right) \right] \cdot \frac{1}{1 + \exp\left(\frac{\mathcal{E}}{k_B \cdot T}\right)} d\mathcal{E} =\end{aligned}$$

(effettuando uno sviluppo di Taylor per $\kappa \sim 0$):

$$= -(\epsilon\chi')' - \frac{N_0 q^2 \chi}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} \exp\left(-\frac{(\mathcal{E} + q\varphi)^2}{2\sigma^2}\right) \frac{\mathcal{E} + q\varphi}{2\sigma^2} \frac{1}{1 + \exp\left(\frac{\mathcal{E}}{k_B \cdot T}\right)} d\mathcal{E}$$

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

Il sistema diventa:

$$\left\{ \begin{array}{l} - \left(\epsilon \left(\delta \varphi^{(k)} \right)' \right) - \frac{d\rho}{d\varphi} \left(\varphi^{(k)} \right) \cdot \delta \varphi^{(k)} = \left(\epsilon \left(\varphi^{(k)} \right)' \right)' + \rho \left(\varphi^{(k)} \right) \\ \varphi^{(k+1)} = \varphi^{(k)} + \delta \varphi^{(k)} \\ \delta \varphi^{(k)}(-t_{semic}) = 0 \\ \delta \varphi^{(k)}(t_{ins}) = 0 . \end{array} \right.$$

per ogni iterazione k fino a convergenza, valutata sulla norma L^∞ dell'incremento.

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

Il sistema diventa:

$$\left\{ \begin{array}{l} - \left(\epsilon \left(\delta \varphi^{(k)} \right)' \right) - \frac{d\rho}{d\varphi} \left(\varphi^{(k)} \right) \cdot \delta \varphi^{(k)} = \left(\epsilon \left(\varphi^{(k)} \right)' \right)' + \rho \left(\varphi^{(k)} \right) \\ \varphi^{(k+1)} = \varphi^{(k)} + \delta \varphi^{(k)} \\ \delta \varphi^{(k)}(-t_{semic}) = 0 \\ \delta \varphi^{(k)}(t_{ins}) = 0 . \end{array} \right.$$

per ogni iterazione k fino a convergenza, valutata sulla norma L^∞ dell'incremento.

È un'equazione integro-differenziale (diffusione-reazione) **lineare**, che dovrà essere discretizzata.

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

Discretizzazione effettuata ai volumi finiti:
metodo **BIM** (**B**ox **I**ntegration **M**ethod).

Spesso utile per equazioni in forma conservativa: $\frac{\partial u}{\partial t} + \nabla \cdot \vec{F}(u) = s(u)$
Si compone delle seguenti fasi:

1. creazione dei box;
2. scrittura del problema in forma integrale locale su ogni singolo box;
3. ipotesi di flusso costante e assemblaggio finale.

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

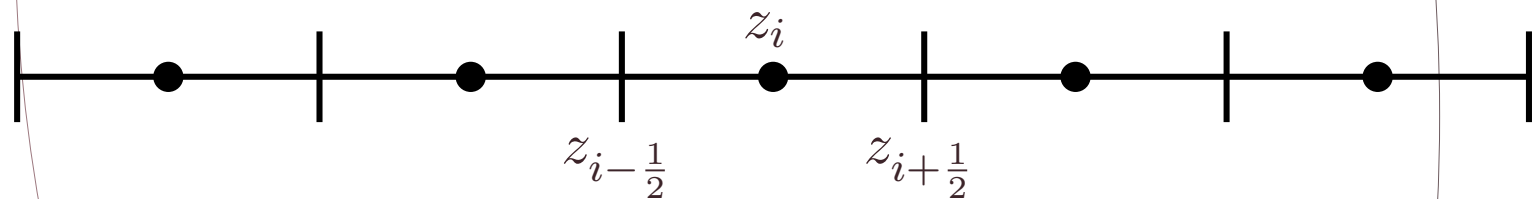
Implementazione

Risultati

■ Fase 1: Creazione dei box

n sotto-domini di lunghezza $h = \frac{1}{n}$

$$\{\mathcal{B}_i\}_{i=1}^n, \mathcal{B}_i = \left(z_{i-\frac{1}{2}}, z_{i+\frac{1}{2}}\right)$$



I volumi finiti approssimano la media su ogni box:

$$(u)_i \approx \frac{1}{h} \int_{z_{i-\frac{1}{2}}}^{z_{i+\frac{1}{2}}} u(z) \, dz$$

■ Fase 2: Formulazione integrale locale

siano $f^{(k)} = \frac{d\rho}{d\varphi}(\varphi^{(k)})$ e $s^{(k)}$ il right-hand side dell'equazione

Integrando sull' i -esimo box:

$$-\int_{z_{i-\frac{1}{2}}}^{z_{i+\frac{1}{2}}} \left(\epsilon \left(\delta\varphi^{(k)} \right)' \right)' dz - \int_{z_{i-\frac{1}{2}}}^{z_{i+\frac{1}{2}}} f^{(k)} \delta\varphi^{(k)} dz = \int_{z_{i-\frac{1}{2}}}^{z_{i+\frac{1}{2}}} s^{(k)} dz .$$

Indicando per semplicità con $F_{i\pm\frac{1}{2}} = -\epsilon(z_{i\pm\frac{1}{2}}) (\delta\varphi^{(k)})'(z_{i\pm\frac{1}{2}})$ l'approssimazione numerica del flusso attraverso i bordi $z_{i-\frac{1}{2}}$ e $z_{i+\frac{1}{2}}$ del box e dividendo per h :

$$\frac{F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}}}{h} - \left(f^{(k)} \delta\varphi^{(k)} \right)_i = \left(s^{(k)} \right)_i .$$

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

■ Fase 3: Assemblaggio finale

problema: $\delta\varphi^{(k)}$ compare ancora sotto il segno di derivata

Sono state esplicitate delle relazioni tra la derivata e i valori “nodali” secondo lo schema di Scharfetter-Gummel.

■ Fase 3: Assemblaggio finale

problema: $\delta\varphi^{(k)}$ compare ancora sotto il segno di derivata

Sono state esplicitate delle relazioni tra la derivata e i valori “nodali” secondo lo schema di Scharfetter-Gummel.

Si giunge alla formulazione algebrica:

$$\begin{cases} \left(\mathcal{K} - \mathcal{M} \cdot \text{diag} \left(d\vec{\rho}^{(k)} \right) \right) \delta\vec{\varphi}^{(k)} = \mathcal{K} \cdot \vec{\varphi}^{(k)} - \mathcal{M} \cdot \text{diag} \left(\vec{\rho}^{(k)} \right) \\ \vec{\varphi}^{(k+1)} = \vec{\varphi}^{(k)} + \delta\vec{\varphi}^{(k)} \end{cases}$$

dove:

- \mathcal{K} matrice di stiffness;
- \mathcal{M} matrice di massa (lumped).

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

Dopo aver risolto il sistema:

- si calcola la carica totale;
- si calcola la capacità elettrica totale.

Questo avviene all'interno di un loop in cui la tensione esterna V_G varia.

Dopo aver risolto il sistema:

- si calcola la carica totale;
- si calcola la capacità elettrica totale.

Questo avviene all'interno di un loop in cui la tensione esterna V_G varia.

Post-processing:

- si confrontano le curve $C(V_G)$ e $\frac{dC}{dV_G}(V_G)$ sperimentali e simulate;
- si “allineano” i picchi della $\frac{dC}{dV_G}(V_G)$: V_{shift} dipende da fenomeni non quantificabili (dipoli permanenti, cariche residue etc.):

$$V_{shift} = \arg \max_{V_g} \frac{dC_{sim}}{dV_g} (V_g; \tilde{X}) - \arg \max_{V_g} \frac{dC_{exp}}{dV_g} (V_g)$$

$$V_g \longleftrightarrow V_g + V_{shift}$$

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

Problemi di modellazione:

- V_{shift} non quantificabile “a priori”;
- \exists capacità parassite (accoppiamento tra i contatti, linee di campo etc.)
→ capacità C_{sb} idealmente connessa in parallelo al MIS;
- misura dello spessore t_{semic} soggetta a errori sperimentali.

Problemi di modellazione:

- V_{shift} non quantificabile “a priori”;
- \exists capacità parassite (accoppiamento tra i contatti, linee di campo etc.)
→ capacità C_{sb} idealmente connessa in parallelo al MIS;
- misura dello spessore t_{semic} soggetta a errori sperimentali.

→ algoritmo di **ottimizzazione**:

1. **Step 1:** si individua la $\sigma^{(k+1)}$ ottima in un range discreto di valori.
2. **Step 2:**

$$C_{sb}^{(k+1)} = C_{sb}^{(k)} + C_{exp}(V_{g,max}) - C_{sim} \left(V_{g,max}; \left[C_{sb}^{(k)}, t_{semic}^{(k)}, \sigma^{(k+1)} \right] \right)$$

3. **Step 3:**

$$t_{semic}^{(k+1)} = \epsilon_{semic} \left(\frac{1}{C_{exp}(V_{g,min}) - C_{sb}^{(k+1)}} - \frac{t_{ins}}{\epsilon_{ins}} \right)$$

Nel sistema lineare i termini integrali $\vec{\rho}$ e $d\vec{\rho}$ devono essere approssimati.
→ formule di **quadratura**:

■ Gauss-Hermite:

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

■ Gauss-Laguerre:

$$n(\varphi) = \frac{N_0}{\lambda} \int_0^{+\infty} e^{-\alpha} \left(1 + \exp \left(\frac{\lambda\alpha - q\varphi}{k_B \cdot T} \right) \right)^{-1} d\alpha$$

$$\Rightarrow \int g(z)f(z) dz \approx \sum_{i=1}^{N_g} w_i \cdot f(z_i)$$

e analogamente per la derivata ($d\vec{\rho}$).

Implementati metodo **diretto** e **iterativo** per il calcolo di pesi e nodi.

Introduzione

Modello

Metodi numerici

Linearizzazione

Discretizzazione

Post-processing

Fitting

Regole di quadratura

Sistemi lineari

Implementazione

Risultati

Matrice del sistema **simmetrica, definita positiva**

→ fattorizzazione di **Cholesky** $A = LDL^T$

L triangolare inferiore e D diagonale

$A\vec{w} = \vec{b}$ diventa:

$$i) \quad L\vec{x} = \vec{b}$$

$$ii) \quad D\vec{y} = \vec{x}$$

$$iii) \quad L^T\vec{w} = \vec{y}$$

In alternativa: **Preconditioned Conjugate Gradient**. Un buon preconditionatore è basato sulla fattorizzazione LU incompleta.

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

■ Eigen (algebra lineare)

- libreria template (non dev'essere compilata, velocità di esecuzione e portabilità, ma tempi di compilazione maggiori e code-bloat)
- matrici e vettori ad allocazione dinamica
- supporto per formato denso e sparso

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

- Eigen (algebra lineare)
 - libreria template (non dev'essere compilata, velocità di esecuzione e portabilità, ma tempi di compilazione maggiori e code-bloat)
 - matrici e vettori ad allocazione dinamica
 - supporto per formato denso e sparso
- GetPot (parsing)
 - singolo header file
 - parsing file di configurazione (parametri numerici, file di input)
 - parsing linea di comando (path del file di configurazione)

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

- Eigen (algebra lineare)
 - libreria template (non dev'essere compilata, velocità di esecuzione e portabilità, ma tempi di compilazione maggiori e code-bloat)
 - matrici e vettori ad allocazione dinamica
 - supporto per formato denso e sparso
- GetPot (parsing)
 - singolo header file
 - parsing file di configurazione (parametri numerici, file di input)
 - parsing linea di comando (path del file di configurazione)
- OpenMP (calcolo parallelo)
 - no message passing (memoria condivisa)
 - “facile” da implementare
 - una errata dichiarazione delle variabili può portare a errori come “Segmentation fault.”

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

- CMake (configurazione e gestione delle dipendenze)
 - portabilità (anche verso sistemi operativi non Unix-like)
 - facile trovare dipendenze nel sistema
 - facile definizione dei target (libreria dinamica, install, doc etc.)

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

- CMake (configurazione e gestione delle dipendenze)
 - portabilità (anche verso sistemi operativi non Unix-like)
 - facile trovare dipendenze nel sistema
 - facile definizione dei target (libreria dinamica, install, doc etc.)
- Gnuplot (generazione di plot)
 - interfaccia `gnuplot-iostream` per C++ (richiede le Boost)
 - sintassi intuitiva, facile importare file `.csv`

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

- CMake (configurazione e gestione delle dipendenze)
 - portabilità (anche verso sistemi operativi non Unix-like)
 - facile trovare dipendenze nel sistema
 - facile definizione dei target (libreria dinamica, install, doc etc.)
- Gnuplot (generazione di plot)
 - interfaccia `gnuplot-iostream` per C++ (richiede le Boost)
 - sintassi intuitiva, facile importare file `.csv`
- Valgrind e gdb (profiling e debugging)
 - Callgrind: chiamata a metodi *getter* lenta
 - restituzione per `const &`
 - dichiarazione classi `friend`
 - Memcheck: nessun comportamento anomalo
 - gdb: “Segmentation fault.” risolto con `backtrace`

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

- CMake (configurazione e gestione delle dipendenze)
 - portabilità (anche verso sistemi operativi non Unix-like)
 - facile trovare dipendenze nel sistema
 - facile definizione dei target (libreria dinamica, install, doc etc.)
- Gnuplot (generazione di plot)
 - interfaccia `gnuplot-iostream` per C++ (richiede le Boost)
 - sintassi intuitiva, facile importare file `.csv`
- Valgrind e gdb (profiling e debugging)
 - Callgrind: chiamata a metodi *getter* lenta
 - restituzione per `const &`
 - dichiarazione classi `friend`
 - Memcheck: nessun comportamento anomalo
 - gdb: “Segmentation fault.” risolto con `backtrace`
- git (sistema di controllo di versione)
- Doxygen (documentazione)
- Artistic style (formattazione)

La classe CsvParser

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

```
1  CsvParser() = delete;  
2  CsvParser(const std::string &, const bool & = true);  
3  
4  RowVectorXr importRow(const Index &);  
5  MatrixXr    importRows(const std::initializer_list<Index> &);  
6  ...  
7  VectorXr    importCol(const Index &);  
8  MatrixXr    importCols(const std::initializer_list<Index> &);  
9  ...  
10 Real        importCell(const Index &, const Index &);  
11 MatrixXr    importAll();
```

Importa .csv (separati da virgola, tab, due punti o spazio) in strutture dati delle Eigen: legge i file contenenti i parametri da simulare e i dati sperimentali

La classe ParamList

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

SIM	SPESSORE SC [m]	SPESSORE INS [m]	EPSILON_R SC [-]	EPSILON_R INS [-]	T [K]	Wf [V]	Ea [V]	N0 [m]
1	6.36E-008	0.000000476	2.9	2.82	295	5	3	
2	6.36E-008	0.000000476	2.9	2.82	295	5	3.2	
3	6.36E-008	0.000000476	2.9	2.82	295	5	3.4	
4	6.36E-008	0.000000476	2.9	2.82	295	5	3.6	
5	6.36E-008	0.000000476	2.9	2.82	295	5	3.8	
6	6.36E-008	0.000000476	2.9	2.82	295	5	4	

```
1 friend class GaussianCharge ;
2 friend class ExponentialCharge ;
3 friend class DosModel ;
4
5 ParamList() = default ;
6 explicit ParamList(const RowVectorXr &) ;
7 ...
```

Gestisce i parametri della simulazione (parametri fisici, range di tensioni etc.).

Fornisce metodi *getter/setter* per leggerli/modificarli

La classe QuadratureRule

Introduzione

Modello

Metodi numerici

Implementazione

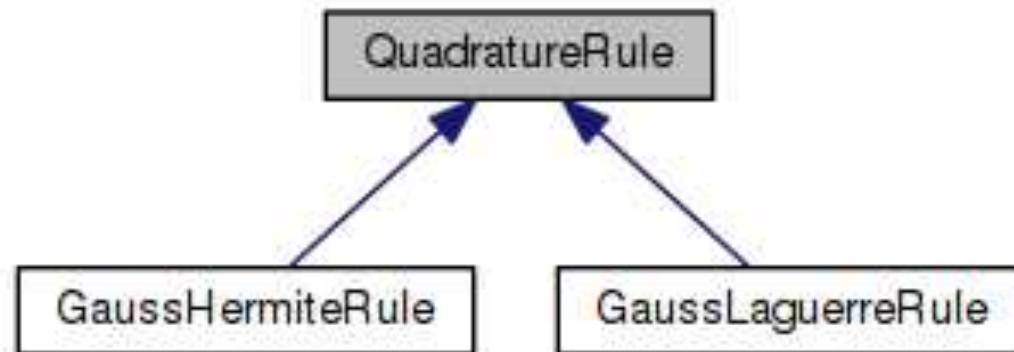
Strumenti utilizzati

Casi test

Risultati

```
1 QuadratureRule() = delete;  
2 QuadratureRule(const Index &);  
3  
4 virtual void apply() = 0;  
5 ...  
6  
7 void apply_iterative_algorithm(const Index &, const Real &);  
8 void apply_using_eigendecomposition();
```

Implementa gli algoritmi (diretto e iterativo) per calcolare nodi e pesi.

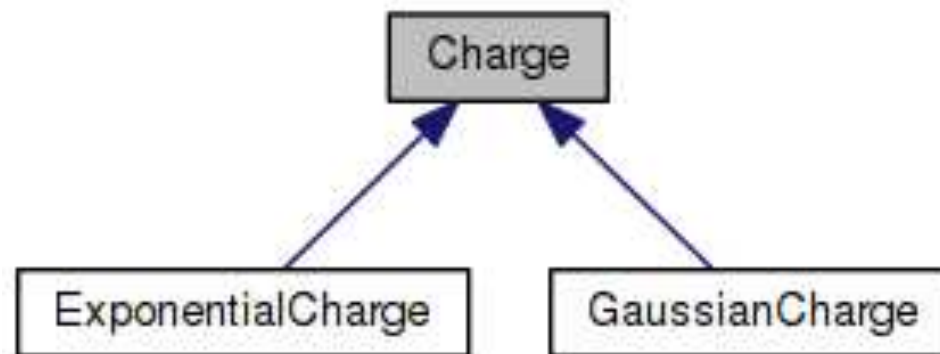


La classe Charge

- Introduzione
- Modello
- Metodi numerici
- Implementazione
 - Strumenti utilizzati
 - Casi test
- Risultati

```
1 Charge() = delete;  
2 Charge(const ParamList &, const QuadratureRule &);  
3  
4 virtual VectorXr charge(const VectorXr &) const = 0;  
5 virtual VectorXr dcharge(const VectorXr &) const = 0;
```

Utilizzando le formule di quadratura, assembla i termini $\vec{\rho}$ e $d\vec{\rho}$.



Le abstract factory

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

```
1 class ChargeFactory {  
2     virtual Charge * BuildCharge(const ParamList &, const  
3         QuadratureRule &) = 0;  
};
```

```
1 class QuadratureRuleFactory {  
2     virtual QuadratureRule * BuildRule(const Index &) = 0;  
3 };
```



(a) ChargeFactory

(b) QuadratureRuleFactory

Design Pattern *creazionale*

Introduzione

Modello

Metodi numerici

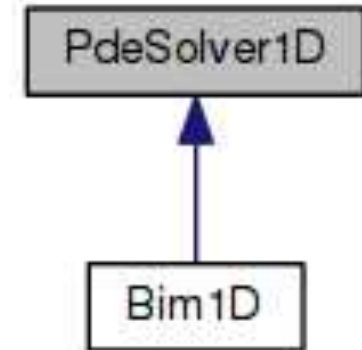
Implementazione

Strumenti utilizzati

Casi test

Risultati

```
1 class PdeSolver1D {
2     PdeSolver1D() = delete;
3     PdeSolver1D(VectorXr &);
4
5     virtual void assembleAdvDiff(...) = 0;
6     virtual void assembleStiff(...) = 0;
7     virtual void assembleMass(...) = 0;
8     ...
9 };
```



```
1 class NonLinearPoisson1D {
2     NonLinearPoisson1D(const PdeSolver1D &, const Index &, const
3         Real &);
4     void apply(const VectorXr &, const VectorXr &, const Charge &);
5     ...
6 };
```

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

```
1 explicit DosModel(const ParamList &);  
2  
3 void simulate (...);  
4 void post_process (...);  
5  
6 void save_plot (...) const;  
7 ...
```

Il metodo `simulate()` esegue la simulazione ed effettua il post-processing, che salva in output i risultati:

- `_info.txt`, contenente informazioni sull'andamento della simulazione;
- `_CV.csv`, contenente i valori simulati e sperimentali della curva $C(V_g)$ e della sua derivata;
- `_plot.png`, contenente il plot di confronto;
- `_plot.gp`, lo script Gnuplot con cui è stato ottenuto.

Introduzione

Modello

Metodi numerici

Implementazione

Strumenti utilizzati

Casi test

Risultati

■ `simulate_dos`

1. definiti oggetti `GetPot`;
2. vengono letti i nomi dei file di input, le simulazioni da effettuare, il numero di thread e i nomi delle directory di output;
3. loop parallelo: ogni thread crea un `DosModel`;
4. viene chiamato il metodo `simulate`.

■ `simulate_dos`

1. definiti oggetti `GetPot`;
2. vengono letti i nomi dei file di input, le simulazioni da effettuare, il numero di thread e i nomi delle directory di output;
3. loop parallelo: ogni thread crea un `DosModel`;
4. viene chiamato il metodo `simulate`.

■ `fit_dos`

All'interno del loop di ottimizzazione:

1. viene individuata $\sigma^{(k+1)}$, eseguendo in parallelo tutte le simulazioni al variare del parametro nel range;
2. viene calcolato $C_{sb}^{(k+1)}$;
3. viene calcolato $t_{semic}^{(k+1)}$.

Introduzione

Modello

Metodi numerici

Implementazione

Risultati

Simulazioni

Fitting

Possibili sviluppi

Conclusioni

Condensatore MIS di tipo n , basato su P(NDI2OD-T2).

La forma considerata per la DOS è gaussiana (singola o doppia).

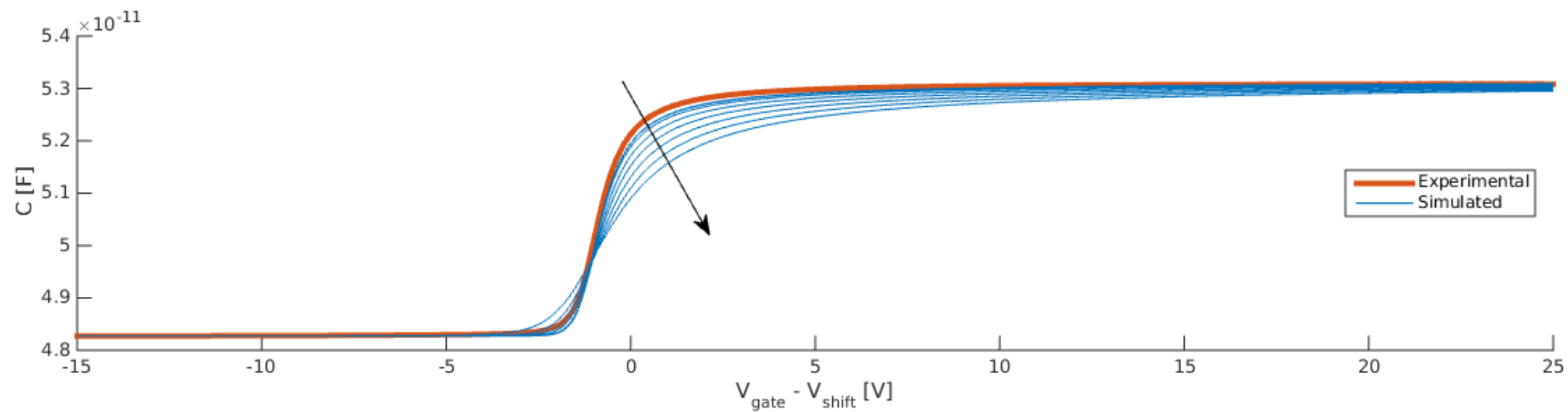
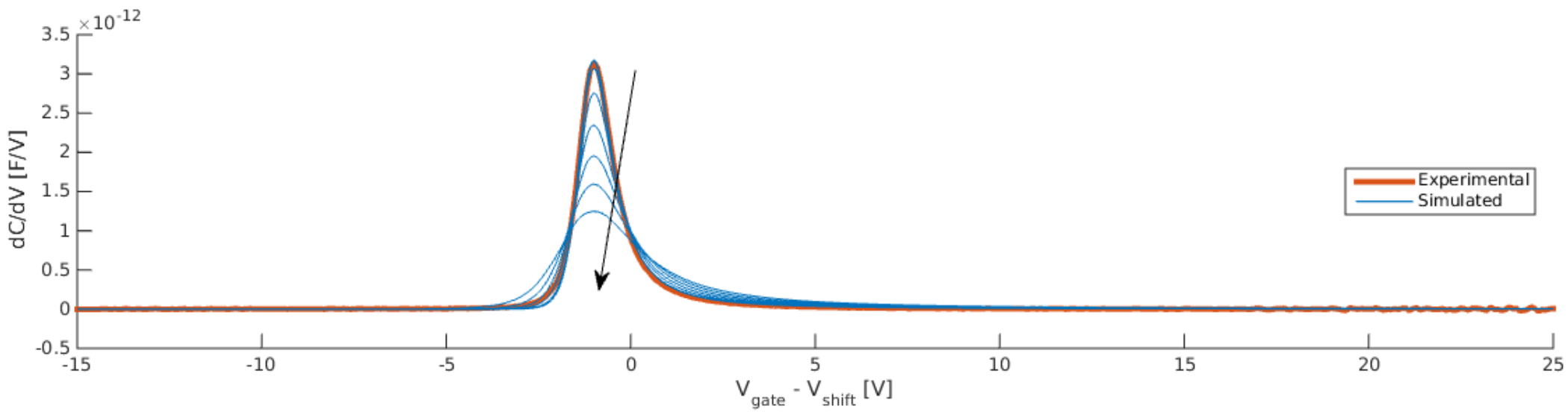
Dati concessi dal **CNST** dell'*Istituto Italiano di Tecnologia*

Dieci gruppi di simulazioni, al variare di:

1. σ
2. N_0
3. Φ_B
4. σ_2
5. $\varphi_s, 2$ (shift della seconda gaussiana)
6. numero di nodi della mesh
7. numero di suddivisioni per V_g
8. range per V_g
9. temperatura T (gaussiana singola)
10. temperatura T (gaussiana doppia)

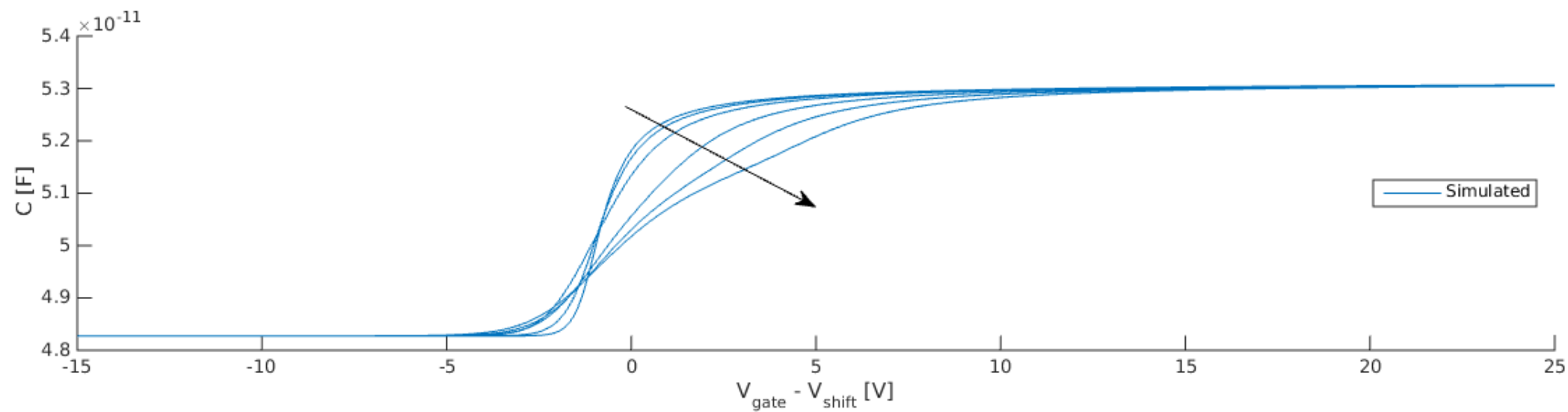
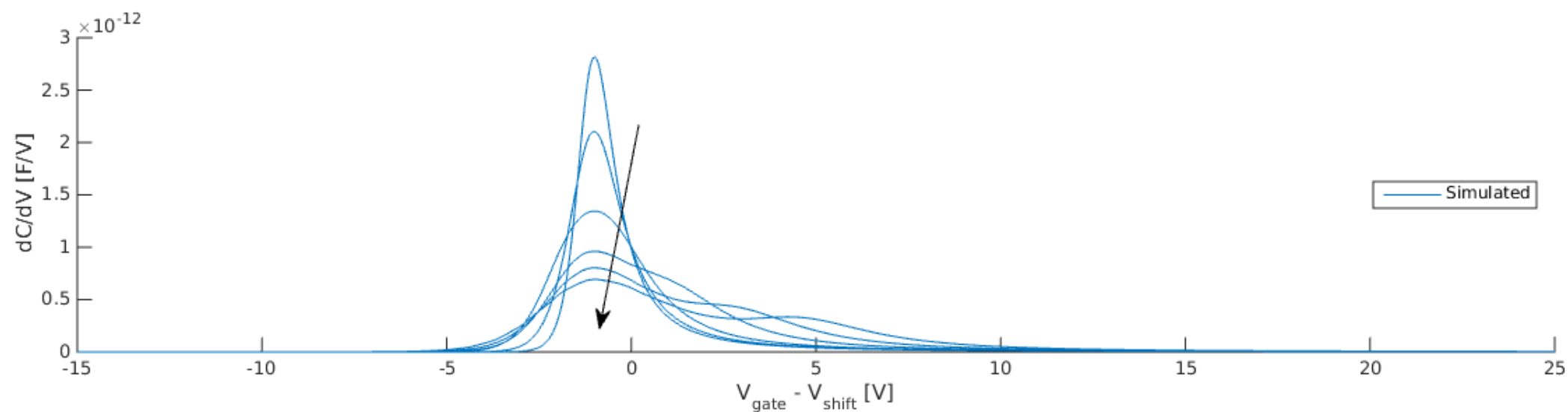
Simulazioni: al variare di σ

$N_0 = 10^{27} m^{-3}$, σ varia da 1 a $10.5 k_B \cdot 300K$



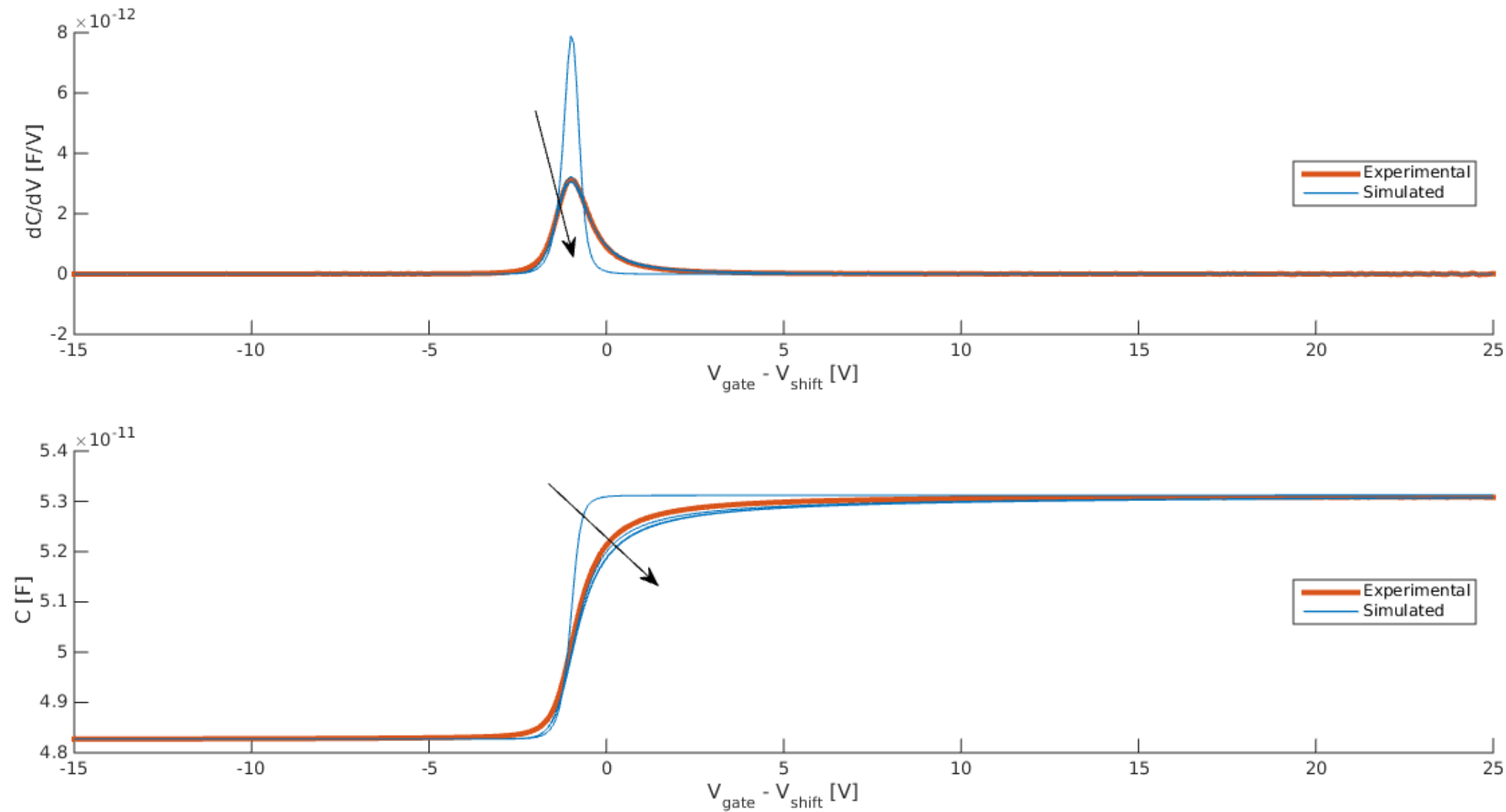
Simulazioni: al variare di σ_2

$N_0 = 10^{27}$, $\sigma = 3$, $N_{0,2} = 10^{24}$, σ_2 varia da 4 a 9.5, $\varphi_{s,2} = 0.1V$



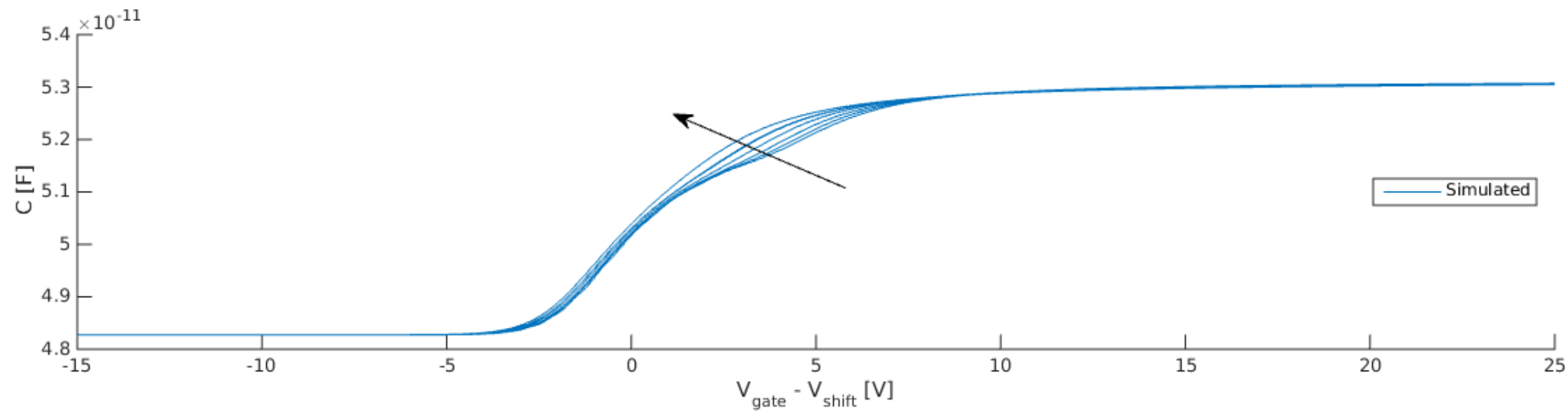
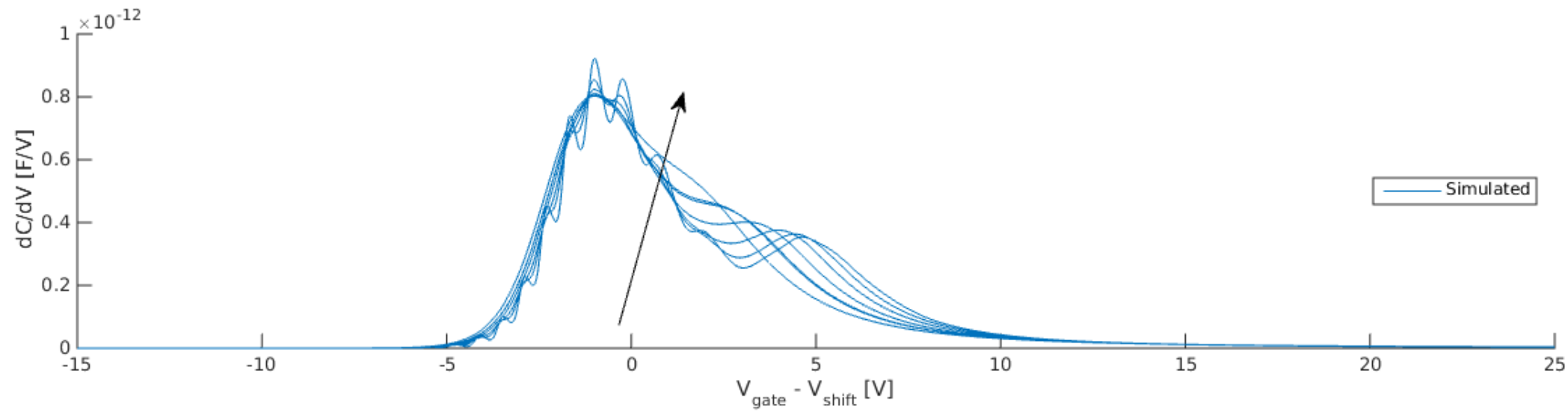
Simulazioni: al variare del range di V_g

$N_0 = 10^{27}$, $\sigma = 3$, la dimensione del range varia da 1000 a 8000 step



Simulazioni: al variare di T

$N_{0,2} = 10^{25}$, $\sigma_2 = 8$, $\varphi_{s,2} = 0.1$, T varia da 100 a 350K



Introduzione

Modello

Metodi numerici

Implementazione

Risultati

Simulazioni

Fitting

Possibili sviluppi

Conclusioni

Eseguito su alcuni casi notevoli. Numero di iterazioni pari a 3.

Simulazioni al variare di σ in $[\sigma - 2, \sigma + 3]$.

C_{sb} e t_{semic} convergono a $1.06649 \cdot 10^{-11} F$ e $63.49 nm$

Eseguito su alcuni casi notevoli. Numero di iterazioni pari a 3.

Simulazioni al variare di σ in $[\sigma - 2, \sigma + 3]$.

C_{sb} e t_{semic} convergono a $1.06649 \cdot 10^{-11} F$ e $63.49 nm$

Norma scelta: H^1

Partendo da $\sigma = 3.5$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	1.5
2	0.8
3	0.1

Partendo da $\sigma = 4.5$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	2.5
2	0.5
3	0.1

Partendo da $\sigma = 5$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	3
2	1
3	1

Partendo da $\sigma = 6$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	4
2	2
3	0.1

Introduzione

Modello

Metodi numerici

Implementazione

Risultati

Simulazioni

Fitting

Possibili sviluppi

Conclusioni

Eseguito su alcuni casi notevoli. Numero di iterazioni pari a 3.

Simulazioni al variare di σ in $[\sigma - 2, \sigma + 3]$.

C_{sb} e t_{semic} convergono a $1.06649 \cdot 10^{-11} F$ e $63.49 nm$

Norma scelta: distanza tra i picchi in $\frac{dC}{dV_g}$

Partendo da $\sigma = 3.5$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	1.5
2	0.1
3	0.1

Partendo da $\sigma = 4.5$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	2.5
2	0.5
3	0.3

Partendo da $\sigma = 5$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	3
2	1
3	0.1

Partendo da $\sigma = 6$:

Iterazione	$\sigma_{opt}[k_B \cdot 300K]$
1	4
2	2
3	0.1

Introduzione

Modello

Metodi numerici

Implementazione

Risultati

Simulazioni

Fitting

Possibili sviluppi

Conclusioni

- Geometria più realistica in **2D** o **3D**;
- regime non quasi-statico o tempo-variante (sistema completo drift-diffusion);
- migliorare l'algoritmo di fitting, formalizzandolo dal punto di vista teorico e numerico;
- **ottimizzazione multi-obiettivo** (forme più complesse per la DOS);
- effetti di adattività di griglia (l'area vicino al "picco" della derivata soggetta a variazioni).

Introduzione

Modello

Metodi numerici

Implementazione

Risultati

Simulazioni

Fitting

Possibili sviluppi

Conclusioni

- Risultati coerenti con Octave (errore $< 0.1\%$)
stesso metodo, diversa implementazione tra Eigen e Cholmod;
- il solutore ha un'influenza trascurabile sia nei tempi di esecuzione che nei risultati (confronti tra Cholesky e BiCGSTAB);
- esecuzione molto più veloce:
8 core (hyperthreading) @ $2.2GHz$:
 - Octave 30 minuti vs. C++ 16 secondi
 - Octave 2+ ore vs. C++ 8 minuti



Grazie per l'attenzione