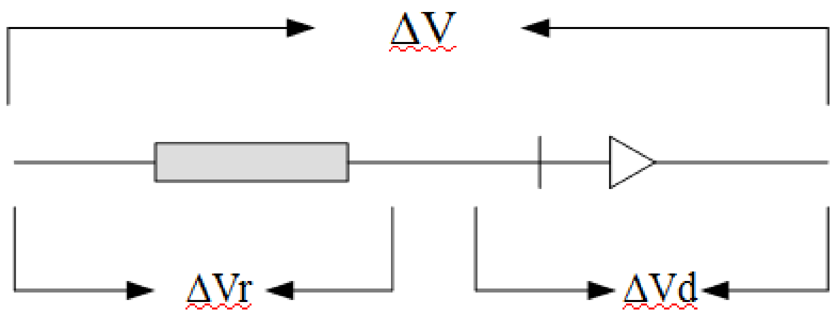
Metodo di fit per il diodo:

Teniamo presente che la tensione ai capi del diodo può essere scomposta in ΔVr e ΔVd come in figura:



ove il rettangolo in grigio rappresenta un corpo resistivo con resistenza pari a quella del diodo reale e il diodo ivi rappresentavo altro non è che un diodo ideale secondo l'equazione di Shockley.
Dunque ci aspettiamo che la tensione V campionata con Arduino ai capi del diodo rispetti la legge:

$$V = \Delta V = \Delta V_d + \Delta V_r = \eta V_t \ln\left(\frac{I + I_0}{I_0}\right) + I \cdot R \quad \text{(legge 1)}$$

All'interno dello script *fit_file.py,* voltages rappresentano le ascisse dei punti campionati, mentre I currents rappresentano le ordinate. I relativi errori adottati entro il fit sono voltageErrs e currentErrs. A questo punto saremo tentati ad effettuare una sorta di fit "inverso" (delle x in funzione delle y). Tuttavia tale operazione provocherebbe il fallimento inevitabile del fit a causa dei valori negativi/nulli che debitamente si riscontreranno all'interno del logaritmo nella legge 1.
Così si è dovuto procedere secondo una legge di natura approssimata, basata sul metodo di Newton. Per spiegarlo, entriamo in dettaglio nel codice. E' stata redatta una funzione:

```
def curr(V, I0, nVt, R):
    v = V;
   for i in range(Nstep):
      a = deriv_errFun(v, I0, nVt, R)
      v = v - errFun(v, V, I0, nVt, R) /a
   return (V – v)/R
```

la quale verrà in seguito passata entro il modulo curve_fit per la corrente in funzione della tensione ai capi del diodo reale. In pratica questa funzione cerca di linearizzare la relazione tra I e V in un intorno di un determinato V. Come sappiamo infatti dal metodo di Newton, se chiamo f(x) una funzione tale che f(x) = 0 e dato un valore iniziale tale che f(x[0]) = a generico, se procediamo con un numero di iterazioni sufficienti, sappiamo che la relazione di ricorsione che segue ci fornirà un risultato soddisfacente per la nostra x:

$$x[0] = a$$

$$x[N+1] = x[N] - \frac{f(x[N])}{f'(x[N])}$$

ove fra parentesi quadre è indicato il livello di iterazione raggiunto per la x. La x in questione per noi sarà la v tale che:

$$I_0 \cdot e^{\frac{V}{\eta V_t}} = I(V) = \frac{v - V}{R} \quad (2)$$

che in pratica identifica un tratto lineare caratterizzato da una retta con coefficiente angolare negativo pari a -1/R ed un intercetta di I = v/R sull'asse delle y, che come è facile dedurre altro non è che la retta di carico del diodo. Facendo così stiamo quindi cercando il punto di lavoro v per cui la corrente può essere scritta come (v-V)/R (corrente di lavoro), come si evince dal return nella funzione di fit curr(V, parametri liberi).
Sulla base di ciò è facile dedurre che le altre due funzioni che vengono richiamate dentro curr possono essere scritte nella forma:

```
def errFun(V, V0, I0, nVt, R):
   return sck(V, I0, nVt) + (V – V0)/R

def sck(V, I0, nVt):
   return I0*(pylab.exp(V/nVt) – 1

def deriv_errFun(V, I0, nVt, R):
   return I0 / nVt * pylab.exp(V/nVt) + 1./R
```

ove errFun e deriv_errFun altro non sono che la relazione da minimizzare specificata dalla legge 2 e la rispettiva derivata. Quindi il fit è stato effettuato secondo le consuete modalità adottando in primo luogo curr come funzione di fit e currentErrs come errori sui punti (gli errori sulle y). In seguito, sono stati propagati anche gli errori voltageErrs relativi alle x sulle y secondo la consueta modalità.