Carica/scarica condensatore con Arduino

francesco.fuso@unipi.it; http://www.df.unipi.it/∼fuso/dida

(Dated: version 8 - Lara Palla e Francesco Fuso, 3 novembre 2016)

In questa nota si descrive brevemente l'esperienza pratica sull'acquisizione automatizzata dei dati via Arduino applicata alla carica/scarica di un condensatore, mettendo in luce la strategia di misura e le tecniche di trattamento dati, di cui si presentano diverse varianti.

I. INTRODUZIONE

Nell'esperienza si vuole registrare l'andamento temporale della d.d.p. $V(t)$ ai capi di un condensatore che viene caricato e scaricato attraverso un resistore esterno, di resistenza R conosciuta. L'andamento atteso segue le leggi, ben note e semplicissime da determinare, $V(t) = V_0(1 - \exp(-t/\tau))$ e $V(t) = V_0 \exp(-t/\tau)$ per le fasi rispettivamente di carica e scarica; nelle equazioni abbiamo indicato con V_0 la d.d.p. fornita dal generatore che esegue la carica, abbiamo supposto che l'istante iniziale fosse $t = 0$ per entrambi le fasi, e posto $\tau = RC$, con C capacità del condensatore. Volendo, l'esperienza costituisce un modo per determinare il valore della capacità C a partire da quella, tipicamente nota con buona accuratezza attraverso misura con multimetro, di R.

Naturalmente, come vedremo in futuro, esistono altri modi per determinare C, per esempio attraverso l'acquisizione della curva di risposta del filtro RC corrispondente (passa-basso o passa-alto), oppure la misura dell'impedenza (modulo e sfasamento) in funzione della frequenza, o ancora tramite confronto con una capacità di riferimento in un ponte di de Sauty. Tuttavia la registrazione diretta delle curve di carica e scarica è sicuramente interessante, dato che rappresenta l'evidenza sperimentale della soluzione di una classe di famose equazioni differenziali, e fa sempre piacere verificare quanto le previsioni matematiche siano (più o meno) ben riprodotte dalla realtà sperimentale.

L'uso di Arduino consente di registrare per punti e *in modo automatico* le coppie di dati V_j e t_j, che si riferiscono alla d.d.p. ai capi del condensatore e al tempo trascorso nella fase di carica o di scarica, in modo da permettere l'analisi dei dati tramite best-fit, finalizzata essenzialmente a determinare il valore del tempo caratteristico τ e degli altri parametri rilevanti. Come chiariremo nel seguito, esistono delle limitazioni legate soprattutto alle caratteristiche di Arduino e della strategia di acquisizione, che suggeriscono di operare con $\tau \sim 5 - 50$ ms, ottenibile con opportune scelte dei valori di R e C; inoltre il funzionamento di Arduino nelle condizioni dell'esperienza può dare luogo a effetti sistematici che affliggono la ricostruzione degli andamenti.

II. CONFIGURAZIONE DI MISURA

La configurazione concettuale di misura è rappresentata in Fig. 1(a): a un dato istante lo switch viene commutato sulla posizione 1 e il condensatore C, che supponiamo precedentemente scarico, viene caricato dal generatore attraverso la resistenza R; di conseguenza, la d.d.p. ai suoi capi cresce nel tempo secondo la funzione $V(t) = V_0(1 - \exp(-t/\tau_C))$, fino a giungere a un valore asintotico pari a V_0. A un altro dato istante, il commutatore passa sulla posizione 2 e il condensatore, caricato nella fase precedente, si scarica attraverso la resistenza R; di conseguenza, la d.d.p. ai suoi capi diminuisce esponenzialmente nel tempo, secondo la funzione $V(t) = V_0 \exp(-t/\tau_S)$, fino a giungere asintoticamente a zero.

Arduino può fare un sacco di cose (più o meno bene). Come sapete, esso è in grado di eseguire delle misure di d.d.p. intervallate regolarmente nel tempo e di registrarne il valore grazie alla presenza di digitalizzatori, ovvero convertitori analogico/digitali, collegati alle porte di ingresso, per esempio al pin `A0` già impiegato per la misura di tensioni continue. Arduino, inoltre, dispone anche di porte digitali in grado di fornire a comando una tensione (pari nominalmente a V_{ref}, dunque circa 5 V per noi). La porta digitale può, ovviamente, essere azionata via software e dunque la configurazione concettuale di misura di Fig. 1(a) può facilmente essere automatizzata. La Fig. 1(b) mostra lo schema di massima delle connessioni da realizzare con la scheda Arduino: la porta indicata con `A0` (boccola blu) è l'ingresso analogico prescelto per l'esperienza, la porta indicata con `7` (boccola rossa) è l'uscita digitale prescelta per l'esperienza, attivata (accesa) e disattivata (spenta) tramite comando software, `GND` (boccola nera) indica la connessione di massa, o terra, che è ovviamente necessaria per riferire i potenziali in modo corretto allo stesso livello.

Nell'esperienza pratica si fa in modo, secondo quanto descritto nel seguito, di accendere a un dato istante l'uscita digitale collegata al pin `7` e, contemporaneamente (vedi dopo per il significato da dare a questo avverbio), di avviare l'acquisizione della d.d.p. sulla porta analogica collegata al pin `A0`, temporizzata a intervalli nominalmente regolari Δt. Quindi, trascorso un certo tempo,

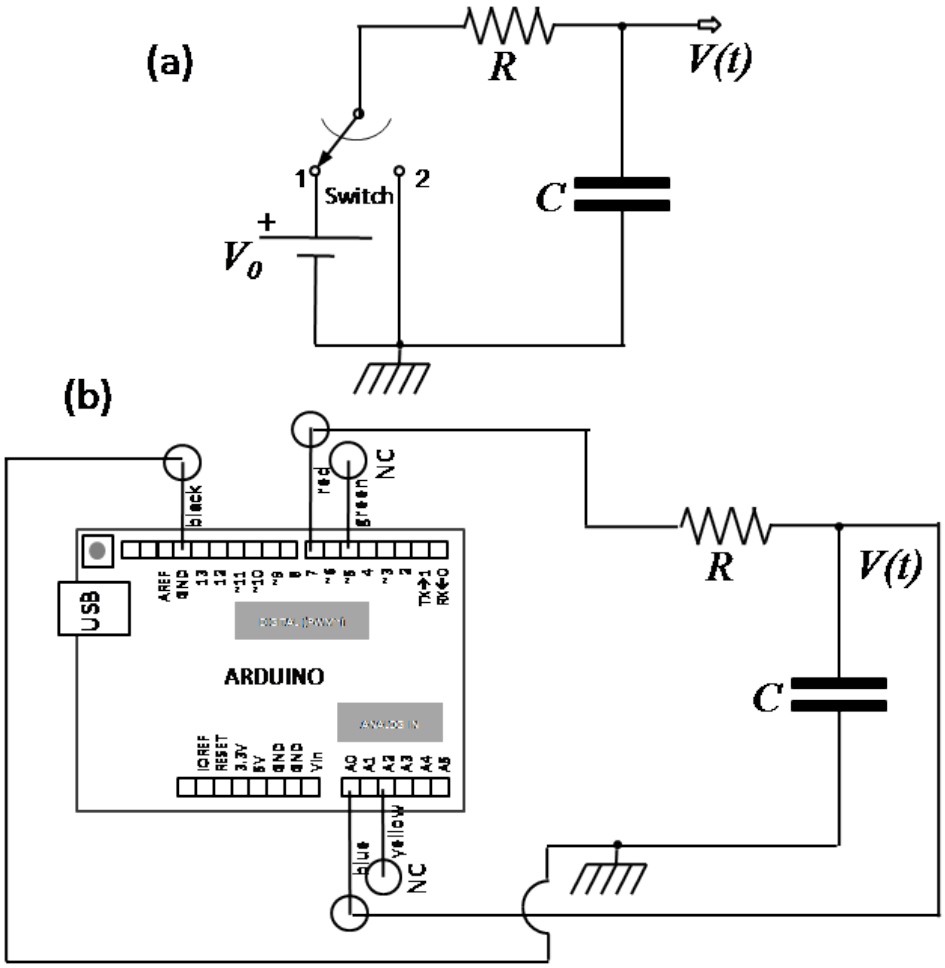

Figura 1. Schema concettuale dell'esperienza (a) e sua realizzazione con Arduino (b). Nel pannello (b) è rappresentata una visione molto schematica e non in scala della scheda Arduino Uno rev. 3 SMD edition usata nell'esperienza. Ci sono cinque collegamenti a altrettanti pin della scheda che terminano con boccole volanti di diverso colore, secondo quanto indicato in figura: solo tre boccole devono essere collegate (NC significa non collegato).

si spegne la porta digitale, e si ricomincia l'acquisizione temporizzata per seguire la fase di scarica. Al termine del processo sono registrati sul computer *due* files che si riferiscono rispettivamente alla fase di carica e a quella di scarica (i nomi sono opportunamente differenziati), entrambi composti da due colonne: la prima riporta i tempi t_j in unità di microsecondi, a partire da un tempo zero nominalmente corrispondente all'inizio della fase di carica, o di scarica, e la seconda il valore digitalizzato V_j della d.d.p., in unità arbitrarie di digitalizzazione, o *digit* secondo la nostra nomenclatura. Di norma, i files sono composti da 256 coppie di dati, ma esiste la possibilità di eseguire cicli di acquisizione che portano a files contenenti un multiplo intero di 256 coppie di dati. Questi files devono poi essere analizzati tramite grafico, best-fit, etc., come illustrato nel seguito per alcuni esempi.

Prima di procedere, è bene sottolineare da subito un paio di differenze tra quanto ci proponiamo di fare e quanto, invece, è schematizzato nell'esperimento concettuale [Fig. 1(a)]: concettualmente abbiamo implicitamente supposto di avere un generatore ideale (nessuna resistenza interna è stata rappresentata nello schema), mentre nella realtà il generatore costituito dalla porta digitale di Arduino ha una resistenza interna piccola, ma non necessariamente trascurabile [1]. D'altra parte, nella fase di scarica vorremmo che la serie RC fosse una maglia, come si realizza commutando lo switch di Fig. 1(a) nella posizione 2. Invece tutto quello che possiamo fare con Arduino è di porre l'uscita digitale 7 a livello basso, cioè "zero". In elettronica, avere un potenziale nullo non implica necessariamente un collegamento "reale" (fisico) con la linea di massa. In altre parole, tra l'uscita digitale 7 posta a potenziale nullo e la massa potrebbe trovarsi una resistenza interna incognita. Queste resistenze interne potrebbero modificare la valutazione del tempo caratteristico τ e, in particolare, dare luogo a un tempo caratteristico di carica τ_C diverso da quello di scarica τ_S. Infine, anche la resistenza di ingresso della porta analogica di Arduino potrebbe giocare un ruolo, che tuttavia supponiamo trascurabile, visto l'elevato valore (100

Mohm) riportato nei datasheets.

III. ESECUZIONE DELL'ESPERIENZA E CAMPIONAMENTO

Conosciamo già le modalità generali di impiego di Arduino nelle nostre esperienze e sappiamo che uno specifico programma, detto *sketch*, istruisce Arduino sulle operazioni da compiere, mentre il "controllo" dell'acquisizione è eseguito via porta seriale (USB) attraverso uno script di Python. Lo script provvede in particolare a stabilire l'intervallo nominale di campionamento, ovvero il ritardo Δt tra due digitalizzazioni successive impostato via software, a far partire le misure, e infine a leggere i dati tramite porta seriale, permettendone la registrazione su (due distinti) files nel disco rigido del computer, pronti per ulteriori analisi.

Fondamentalmente, in questo esperimento viene richiesto ad Arduino di compiere operazioni molto simili a quelle dell'esperienza sulla misura di d.d.p. continue, per cui sketch e script sono molto simili e simile è la filosofia di misura, compreso l'uso delle istruzioni di sketch che permettono l'"overclock" del digitalizzatore. La principale differenza è che Arduino deve preoccuparsi di accendere e spegnere la porta digitale di uscita corrispondente al pin `7` all'inizio rispettivamente della fase di carica e di scarica. Inoltre sketch e script sono congegnati in modo da permettere l'acquisizione sequenziale dei dati relativi a carica e scarica del condensatore, che danno luogo a due files distinti contenenti ognuno 256 coppie di dati (tempo e d.d.p. ai capi del condensatore). Come per la misura di d.d.p. continue, anche in questo caso esiste la possibilità di eseguire cicli, che producono files con numero di righe multiplo di 256.

Dal punto di vista concettuale, le misure di questa esperienza sono *sincrone*, e non asincrone come nel caso delle d.d.p. continue: infatti l'acquisizione parte assieme alla modifica (spento-acceso e acceso-spento) della d.d.p. in uscita dalla porta digitale. È evidente che la sincronizzazione è eseguita *via software*, cioè grazie a un'opportuna sequenza di istruzioni nello sketch di Arduino: affronteremo in seguito situazioni in cui la sincronizzazione dovrà avvenire in modo differente, sfruttando, in termini generali, dei segnali di "trigger". Inoltre è ovvio che la sincronizzazione, come tutta la gestione della tempistica di acquisizione, soffre di un'incertezza non nulla, che possiamo convenzionalmente assumere come $\delta t = \pm 4$ μs, sulla base di quanto abbiamo già verificato.

Lo scopo dell'esperienza è quello di ricostruire l'andamento temporale del segnale di carica e scarica del condensatore, che si sviluppano in tempi caratteristici $\tau = RC$. A partire dall'istante $t_0 = 0$ (entro l'incertezza), Arduino *campionerà* il segnale, cioè eseguirà la digitalizzazione del segnale presente al pin `A0`, a intervalli di tempo separati nominalmente di Δt. Questo intervallo è impostabile via software (agendo sullo script di Python) tra 100 e 900 μs, in unità di 100 μs. L'esperienza sulla misura delle d.d.p. continue ci ha insegnato che la digitalizzazione avviene a intervalli temporali quasi-regolari (regolari entro l'incertezza) il cui valore unitario è superiore all'impostazione nominale di Δt per un ulteriore intervallo temporale, che qui chiamiamo Δt_{dig}, necessario affinché la digitalizzazione abbia luogo; Δt_{dig} dipende dalle specifiche condizioni di uso e, probabilmente, dalla specifica scheda impiegata: il suo valore è generalmente compreso tra 10 e 20 μs.

Per ricostruire fedelmente l'andamento temporale del processo sotto analisi, occorre naturalmente campionare con un corretto intervallo temporale. I campionamenti, infatti, sono 256 e coprono quindi un intervallo complessivo $\Delta t_{tot} \simeq 256 \times (\Delta t + \Delta t_{dig})$: in generale, occorre $\Delta t_{tot} \geq \zeta\tau$, con $\zeta > 1$. La scelta del parametro ζ dipende un po' dai gusti: per ricostruire l'intero processo *compresi gli asintoti* occorre $\zeta \sim 4 - 6$, o superiore, ma per avere un'adeguata "sensibilità" nel best-fit è sufficiente $\zeta \sim 2 - 3$ (i dati corrispondenti agli asintoti sono in genere poco significativi nelle routine di minimizzazione).

Dal punto di vista pratico tutto dipende dalla scelta dei valori di R e C, la cui conoscenza (solo in valore nominale, per C, mentre R può essere misurata con il multimetro) stabilisce il valore atteso, τ_{att}, della costante tempo. La scelta deve essere eseguita in maniera tale che sia possibile soddisfare la condizione appena scritta (scegliete voi, ragionevolmente, il valore di ζ) usando gli intervalli di campionamento nominali Δt impostabili via software.

Nel dimensionamento di R e C dovete tenere conto di un'ulteriore limitazione, legata al fatto che l'uscita digitale di Arduino (il pin `7` usato per caricare a comando il condensatore) si comporta come un generatore reale: è opportuno montare nel circuito una resistenza R di valore molto maggiore della resistenza interna di questo generatore, in modo tale che essa risulti trascurabile nella determinazione di τ.

Un altro modo per illustrare la situazione si basa sull'analisi della richiesta di corrente a questa porta, che si consiglia limitare a pochi mA. Nella fase di carica del condensatore, l'intensità di corrente massima richiesta (all'istante iniziale) è $I_{max} = dQ(t)/dt|_{t=0} = CV_0/\tau_C = V_0/R$ (nel modello si trascura evidentemente la resistenza interna del generatore). Poiché nel nostro caso $V_0 \simeq 5$ V, è bene scegliere $R \geq 1$ kohm. Fatta la scelta di R, C può essere selezionato sulla base del Δt prescelto e, naturalmente, della disponibilità dei condensatori.

Dunque dal punto di vista pratico è necessario dimensionare correttamente il circuito e scegliere un Δt ragionevole. Altrimenti si corre il rischio di avere un *sottocampionamento* (i dati sono registrati a intervalli troppo grandi per il processo sotto analisi) o un *sovracampionamento* (i dati sono registrati su un intervallo temporale totale che è troppo breve per il processo sotto analisi). Entrambi sono situazioni da evitare accuratamente, poiché esse possono facilmente condurre a conclusioni erronee.

A. Lo script di Python

Lo script di Python che controlla Arduino è molto simile a quello già impiegato nell'esperienza della misura di tensioni continue, inclusa la possibilità di eseguire in automatico cicli di acquisizioni (si consiglia di servirsene solo a ragion veduta). Ovviamente, a differenza dello script allora impiegato, qui sulla console non vengono riportati i valori di media e deviazione standard sperimentale, che non avrebbero alcun senso.

Le istruzioni che devono essere fornite dall'esterno (modificando opportunamente lo script prima di lanciarlo) sono:

1. la *parte comune* del nome dei files che verranno registrati, da scegliere secondo i gusti. Lo script provvede infatti a registrare *due* distinti files di testo, ognuno fatto di 256 righe e due colonne (tempo t_j in microsecondi, valore della d.d.p. digitalizzata V_j in digit, con j che corre da 0 a 255), che si riferiscono alla fase di carica e a quella di scarica. Detta `pippo` la parte comune del nome dei files, essi avranno nome rispettivamente `pippo_C.txt` e `pippo_S.txt`. Al solito, la directory di default per la loro registrazione nei computer di laboratorio è `../dati_arduino/`.

2. L'intervallo di campionamento *nominale* Δt, ovvero la distanza in tempo tra due misure successive. Questo intervallo deve essere fornito nello script in unità di 100 μs (da 100 a 900 μs).

La struttura e la sintassi delle varie istruzioni che compaiono nello script non presentano particolari differenze rispetto a quanto usato nella costruzione automatizzata del campione di misure di tensioni continue (costanti) e dunque non occorrono molte spiegazioni. L'unico aspetto che può meritare un commento è la modalità con cui vengono creati i due files. La fase di carica e quella di scarica vengono attivate sequenzialmente, e dunque Arduino scrive sulla porta seriale per prime le 256 coppie di dati per la fase di carica, e quindi, dopo averne eseguito la misura, le 256 coppie per la fase di scarica. Un insieme di condizioni nello script permette di indirizzare correttamente questi dati nei due distinti files.

Lo script, che trovate nei computer di laboratorio e anche, per referenza, in rete sotto il nome `cond2016.py`, è riportato con qualche commento qui nel seguito.

```
import serial # libreria per gestione porta seriale (USB)
import time # libreria per temporizzazione
import numpy

nacqs = 1 # numero di acquisizioni da registrare (ognuna da 256x2 coppie di punti)

FileName='pippo'  # parte comune nome file << DA CAMBIARE SECONDO NECESSITA'
Directory = '../dati_arduino/' # directory dei files di dati
FileNameC=(Directory+FileName+'_C.txt') # crea nome file dati carica
FileNameS=(Directory+FileName+'_S.txt') # crea nome file dati scarica
outputFileC = open(FileNameC, "w" ) # apre file dati predisposto per scrittura carica
outputFileS = open(FileNameS, "w" ) # apre file dati predisposto per scrittura scarica

print('Please wait')

for j in range (1,nacqs+1):
    ard=serial.Serial('/dev/ttyACM0',19200)  # apre porta seriale (occhio alla sintassi)
    print('Start Acquisition ',j, ' of ',nacqs) # scrive sulla console (terminale)
    time.sleep(2)   # aspetta due secondi per evitare casini

    ard.write(b'1') # scrive il carattere per l'intervallo di campionamento
                    # in unita' di 100 us << DA CAMBIARE A SECONDA DEI GUSTI
                    # l'istruzione b indica che e' un byte (carattere ASCII)
    time.sleep(2) # aspetta due secondi per evitare casini
    # loop lettura dati carica da seriale (256 coppie di dati: tempo in us, valore digitalizzato di d.d.p.)
    for i in range (0,256):
        data = ard.readline().decode() # legge il dato e lo decodifica
        if data:
            outputFileC.write(data) # scrive i dati sul file
    # loop lettura dati scarica da seriale (256 coppie di dati: tempo in us, valore digitalizzato di d.d.p.
    for i in range (0,256):
        data = ard.readline().decode() # legge il dato e lo decodifica
```

```
        if data:
            outputFileS.write(data) # scrive i dati sul file
    ard.close() # chiude la comunicazione seriale con Arduino
outputFileC.close() # chiude il file dei dati per la carica
outputFileS.close() # chiude il file dei dati per la scarica
print('end') # scrive sulla console che ha finito
```

B. Lo sketch di Arduino

Anche lo sketch di Arduino ha forma e struttura molto, molto simili a quelle dello sketch usato per l'esperienza del campione delle misure costanti. Come di consueto, esso è diviso in tre blocchi: la dichiarazione delle variabili, l'inizializzazione, il ciclo di misure.

Le differenze principali con lo sketch impiegato in precedenza sono, sostanzialmente: (i) la duplicazione dei cicli di campionamento/digitalizzazione e scrittura sulla porta seriale, dovuta alla necessità di registrare i dati di carica e scarica (gli uni di seguito agli altri); (ii) l'accensione e lo spegnimento della porta digitale corrispondente al pin 7 sincronizzata con l'inizio di ogni fase e realizzata ponendo a livello rispettivamente "alto" o "basso" la porta stessa.

Anche in questa esperienza le prime due misure vengono "scartate", cioè non registrate nel file, operazione quanto mai opportuna per prevenire misure erronee. Infatti quando la porta digitale viene accesa o spenta può verificarsi la propagazione di impulsi di corrente (*spikes*) all'interno del microcontroller, che potrebbero facilmente risultare in valori spuri della d.d.p. all'ingresso analogico. Le corrispondenti misure sono sicuramente degli artefatti, che è bene escludere dalla registrazione. Osservate che l'azzeramento del cronometro interno di Arduino avviene, agendo come di consueto sulla variabile `StartTime`, subito dopo che la porta digitale ha cambiato il suo stato. Questo è concettualmente corretto, perché consente di porre lo zero dei tempi (separatamente per le due fasi) in corrispondenza con l'inizio delle due fasi, entro il tempo necessario perché il microcontroller esegua le varie istruzioni software (presumbilmente entro l'incertezza). La conseguenza pratica è che le registrazioni non avranno inizio dal tempo zero, ma a partire da un certo ritardo. Notate che questa circostanza non ha conseguenze per l'analisi di nostro interesse, in particolare per il best-fit.

Infine osservate la presenza, in posizioni strategiche, di cicli di attesa (per esempio `delay(2000);`, l'unità di misura è qui millisecondi) che servono o per evitare intasamenti nel funzionamento della porta seriale o per garantire il raggiungimento pratico delle condizioni asintotiche di carica del condensatore prima che abbia inizio la fase di scarica. Quest'ultima condizione potrebbe non essere completamente soddisfatta se il tempo caratteristico determinato dalla scelta di R e C è particolarmente lungo.

Lo sketch, che è disponibile nei computer di laboratorio e anche in rete, sotto il nome di `cond2016.ino`, è riportato nel seguito con qualche commento.

```
// Blocco dichiarazioni
const int analogPin=0; // Definisice la porta A0 usata per la lettura
const int digitalPin=7; // Definisce la porta 7 usata per la carica
int i; // Definisice la variabile intera i (contatore)
int delays; // Definisce la variabile intera delays
int V[256]; // Definisce l'array intero V1
long t[256]; // Definisice l'array t come intero long
long StartTime; // Definisce il valore StartTime come intero long
int start=0; // Definisce il valore start (usato come flag)
// Istruzioni di inizializzazione
void setup()
  {
   Serial.begin(19200); // Inizializza la porta seriale a 19200 baud
   Serial.flush(); // Pulisce il buffer della porta seriale
   pinMode(digitalPin,OUTPUT); // Definisce digitalPin come output
   digitalWrite(digitalPin,LOW); // e lo pone a valore low
   bitClear(ADCSRA,ADPS0); // Istruzioni necessarie per velocizzare
   bitClear(ADCSRA,ADPS2); // il rate di acquisizione analogica
  }
// Istruzioni del programma
void loop()
  {
```

```
    if (Serial.available() >0) // Controlla se il buffer seriale ha qualcosa
      {
        delays = (Serial.read()-'0')*100; // Legge il byte e lo interpreta come ritardo in us
        Serial.flush(); // Svuota la seriale
start=1; // Pone il flag start a uno
      }
  if(!start) return // Se il flag e' start=0 non esegue le operazioni qui di seguito
                    // altrimenti le fa partire (quindi aspetta di ricevere l'istruzione
                    // di partenza)
    delay(2000); // Aspetta 2000 ms per permettere di partire con condensatore scarico
    digitalWrite(digitalPin,HIGH); // Pone digitalPin_uno a livello alto per la carica
    StartTime=micros(); // Misura il tempo iniziale con l'orologio interno (lettura in us)
    for(i=0;i<2;i++) // Fa un ciclo di due letture a vuoto per "scaricare" l'analogPin
       {
        V[i]=analogRead(analogPin);
       }
    for(i=0;i<256;i++) // Loop per la carica
      {
          t[i]=micros()-StartTime; // Legge il timestamp in us, sottrae lo StartTime e mette il risultato i
          V[i]=analogRead(analogPin); // Legge analogPin e lo mette in array V
          delayMicroseconds(delays); // Aspetta tot us
      }
    for(i=0;i<256;i++) // Loop per la scrittura su porta seriale dei 256 dati della carica
      {
        Serial.print(t[i]); // Scrive t[i]
        Serial.print(" "); // Mette uno spazio
        Serial.println(V[i]); // Scrive V[i] e va a capo
      }
   delay(1000); // Aspetta 1000 ms per completare la carica
   digitalWrite(digitalPin,LOW); // Pone digitalPin a livello basso per la scarica
   StartTime=micros(); // Misura il tempo iniziale con l'orologio interno (lettura in us)
   for(i=0;i<2;i++) // Fa un ciclo di due letture a vuoto per "scaricare" l'analogPin
       {
        V[i]=analogRead(analogPin);
       }
    for(i=0;i<256;i++) // Loop per la carica
      {
          t[i]=micros()-StartTime; // Legge il timestamp in us, sottrae lo StartTime e mette il risultato i
          V[i]=analogRead(analogPin); // Legge analogPin e lo mette in array V
          delayMicroseconds(delays); // Aspetta tot us
      }
    for(i=0;i<256;i++) // Loop per la scrittura su porta seriale dei 256 dati della carica
      {
        Serial.print(t[i]); // Scrive t[i]
        Serial.print(" "); // Mette uno spazio
        Serial.println(V [i]); // Scrive V[i] e va a capo
      }
  start=0; // Annulla il flag
    Serial.flush(); // Pulisce il buffer della porta seriale (si sa mai)
}
```

IV. MISURE E BEST-FIT

Prima di discutere un esempio di misure, soffermiamoci su un aspetto generale. Lo scopo dell'esperienza è quello di ricostruire l'*andamento* dei processi di carica e scarica, finalizzato in particolare alla valutazione tramite best-fit dei parametri τ_C e τ_S. Per questi scopi è evidentemente *inutile* convertire la lettura digitalizzata V_j fatta da Arduino in unità fisiche.

Questa affermazione è ovvia nel caso in cui si decida di

impiegare la calibrazione "alternativa", che prevede una relazione di proporzionalità diretta tra grandezza digitalizzata e d.d.p. in unità fisiche. Le equazioni modello che descrivono carica e scarica del condensatore sono infatti proporzionali al parametro V_0 e gli andamenti temporali di nostro interesse non dipendono dal valore che questo termine assume, in particolare se esso è misurato in unità fisiche o in unità arbitrarie di digitalizzazione.

Se, invece, si decide di utilizzare la calibrazione lineare, allora è necessario che le equazioni modello contengano un termine costante, di *offset*, da lasciare come parametro libero del best-fit. Questo termine è di fatto già presente nell'equazione modello della fase carica, e l'ulteriore aggiunta di una costante, che risulterebbe completamente (anti)correlata con gli altri parametri, non avrebbe significato. Invece, nel caso della scarica, può essere opportuno aggiungere all'equazione modello un nuovo termine, che, appunto, tiene conto dell'offset del digitalizzatore.

Le equazioni modello sono, pertanto,

$$V(t) = a(1 - \exp(-t/\tau_C)) \qquad \text{[2 params, charge]} \quad (1)$$
$$V(t) = a\exp(-t/\tau_S) \qquad \text{[2 params, discharge]} \quad (2)$$
$$V(t) = a\exp(-t/\tau_S) + c \qquad \text{[3 params, discharge]} \quad (3)$$

dove $V(t)$ è la d.d.p. ai capi del condensatore digitalizzata da Arduino e i parametri liberi del fit sono a, $\tau_{C,S}$ ed eventualmente c nel caso in cui si decida di impiegare la calibrazione lineare invece di quella proporzionale.

A. Esempio di misura

Per l'esempio qui considerato è stato scelto $R = (7.40 \pm 0.06)$ kohm e $C = 1$ μF (nominale, con tolleranza 20 %, indicata dalla lettera "M" sul corpo del componente). La resistenza impiegata limita la massima corrente richiesta alla porta digitale di Arduino a $V_0/R < 1$ mA, che rende trascurabile il ruolo della resistenza interna della porta digitale di Arduino (R è oltre due ordini di grandezza superiore). Il tempo caratteristico previsto nominalmente è $\tau_{att} \simeq 7.4$ ms, con un'incertezza dominata dalla tolleranza sulla capacità. Avendo a disposizione 256 punti di misura, è stato scelto l'intervallo di campionamento nominale $\Delta t = 100$ μs, che comporta $\zeta \gtrsim 3.8$.

La Fig. 2 mostra i dati (valore digitalizzato V_j in funzione del tempo t_j, convertito in ms per una migliore leggibilità delle scale) in rappresentazione ordinaria. Le incertezze scelte sono quelle "convenzionali": $\delta V_j = \pm 1$ digit e $\delta t_j = \pm 4$ μs. La figura riporta anche le curve di best-fit: per completezza, sono stati eseguiti entrambi i best-fit a due e tre parametri per la fase di scarica. Le Tabb. I e II riportano rispettivamente i risultati dei best-fit e le covarianze normalizzate ottenute. Osservate che nei best-fit di questo esempio è stato tenuto in conto dell'incertezza δt_j attraverso la consueta procedura basata sulla propagazione dell'errore; tuttavia, a parte un'ovvia diminuzione del χ^2, i parametri non mutano il loro valore se questa incertezza viene considerata, o meno. Notate infine che, per la situazione specifica che stiamo considerando in cui tutte le incertezze sono costanti sull'intero set di dati, il best-fit del minimo χ^2, che è stato effettivamente svolto, equivale formalmente a un best-fit dei minimi quadrati.

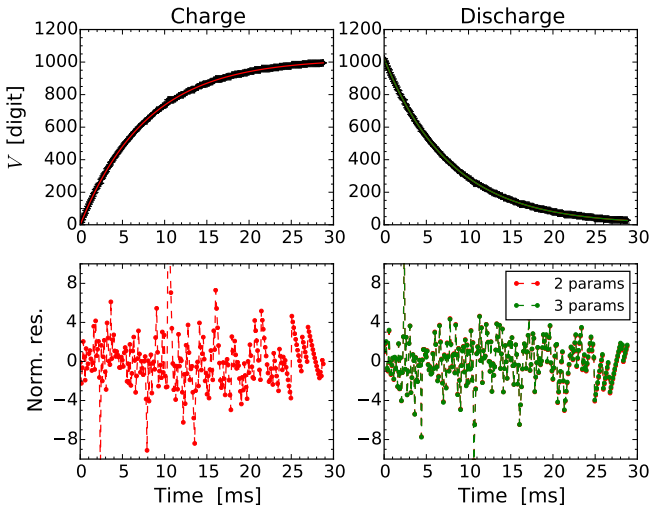

Figura 2. Esempio di misura (carica e scarica, rispettivamente a sinistra e destra di figura) eseguita con la scelta dei valori R e C riportati nel testo; i punti, corredati delle barre di errore, rappresentano dati sperimentali, le linee continue le curve ottenute secondo i best-fit descritti nel testo. I pannelli in basso riportano i grafici dei residui normalizzati.

Senza entrare troppo nello specifico nella discussione, possiamo osservare quanto segue:

- tutti i best-fit realizzati portano a valori di χ^2_{rid} molto maggiori di uno, come ci si poteva aspettare guardando il grafico dei residui normalizzati;
- nei grafici dei residui normalizzati si osservano "picchi" (spikes) e andamenti peculiari (una sorta di "creste" ripetute), che indicano uno scarso accordo tra dati e best-fit;
- l'introduzione del terzo parametro nella funzione modello per la scarica non modifica in maniera sostanziale l'esito dei fit, anche in conseguenza del piccolo valore di c ottenuto per l'esempio specifico [2];
- i valori dei parametri ottenuti dai best-fit suonano generalmente accettabili sulla base di semplici considerazioni fisiche.

In particolare, il parametro a è ragionevole rispetto alle aspettative: esso è prossimo a 1023 digit, che corrisponde al massimo valore digitalizzato dalla scheda, quello che dovrebbe essere asintoticamente raggiunto nella carica e da cui dovrebbe partire la scarica del condensatore. I valori ottenuti per a, tutti compatibili tra loro, non sono tuttavia compatibili con 1023 digit, probabilmente a

Tabella I. Risultati dei best-fit mostrati in Fig. 2: le due righe di valori si riferiscono all'uso dei best-fit a due e tre parametri. In tutti i casi, è stata impiegata l'opzione `absolute_sigma = True`. Il numero di gradi di libertà (ndof) è pari a 254 per i fit a due parametri e a 253 per quello a tre parametri.

Carica				Scarica			
χ^2	a [digits]	τ_C [ms]	c [digits]	χ^2	a [digits]	τ_S [ms]	c [digits]
1456	1021.5 ± 0.2	7.862 ± 0.004	-	1364	1021.5 ± 0.3	7.882 ± 0.003	-
-	-	-	-	1363	1021.5 ± 0.3	7.887 ± 0.005	-0.21 ± 0.18

Tabella II. Covarianze normalizzate per i diversi parametri e i diversi best-fit eseguiti (similmente a Tab. I, le due righe di valori si riferiscono ai best-fit a due e tre parametri).

Carica			Scarica		
cov_{a,τ_C}	$\mathrm{cov}_{a,c}$	$\mathrm{cov}_{\tau_C,c}$	cov_{a,τ_S}	$\mathrm{cov}_{a,c}$	$\mathrm{cov}_{\tau_S,c}$
0.86	-	-	-0.72	-	-
-	-	-	-0.21	-0.18	-0.86

causa della imperfetta sincronia tra zero dei tempi e inizio della fase di carica o scarica. La correlazione tra i parametri è non sempre piccola: il segno delle covarianze normalizzate è coerente con le aspettative, nei fit a due parametri essi risultano fortemente correlati o anticorrelati e l'introduzione del parametro di offset c nella descrizione della fase di scarica introduce una forte anticorrelazione con il parametro a.

A prescindere da questi commenti, i best-fit danno la possibilità di determinare con buona accuratezza τ_C e τ_S. Possiamo osservare che è $\tau_C < \tau_S$, probabilmente a causa delle diverse resistenze interne della porta digitale di Arduino nelle due fasi. Come si vede nelle tabelle, l'accuratezza relativa con cui la routine di best-fit individua questi parametri è migliore rispetto a quella con cui il tester digitale di laboratorio consente di misurare le resistenze. Dunque, se volessimo utilizzare il best-fit per determinare C, scegliendo per esempio quello della fase di scarica modellata con la funzione a due parametri [3], potremmo dedurre da $\tau_S = (7.882 \pm 0.003)$ ms e dalla misura (indipendente) di $R = (7.40 \pm 0.06)$ kohm un valore per la capacità incognita del condensatore $C = (1.065 \pm 0.009)$ μF, con una accuratezza sostanzialmente dettata da quella sulla misura di R eseguita con multimetro digitale.

B. Una rappresentazione migliore

La rappresentazione ordinaria dei dati in Fig. 2 è decisamente poco soddisfacente dal punto di vista visivo: essa, infatti, non permette di giudicare a prima vista la qualità del best-fit e neanche di capire se gli andamenti sono davvero quelli previsti. *Limitandoci alla sola fase di scarica*, possiamo sicuramente guadagnare in termini di evidenza visiva graficando dati e best-fit in *rappresentazione semilogaritmica*. Infatti, trascurando l'eventuale presenza dell'offset (dunque, facendo riferimento al modello con due parametri), la funzione modello assume la forma di una retta con coefficiente angolare negativo, proporzionale a $1/\tau_S$. Questa retta tende a "incurvarsi" a causa dell'offset, verso l'alto o verso il basso a seconda del segno dell'offset stesso.

La Fig. 3 mostra un esempio di dati rappresentati in scala semilogaritmica. In questo caso, allo scopo di evidenziare al meglio alcuni aspetti, si è usato un nuovo set di dati, ottenuto usando la stessa resistenza R di prima, ma scegliendo un condensatore di capacità nominale $C = 0.47$ μF (tolleranza 20 %). Qui il tempo caratteristico previsto nominalmente è $\tau_{att} \simeq 3.5$ ms, che, mantenendo la scelta $\Delta t = 100$ μs, conduce a $\zeta \gtrsim 7.5$. Pertanto in questo caso il grafico mostra un numero relativamente elevato di dati corrispondenti all'asintoto, con valori digitalizzati prossimi allo zero. Per esigenze di spazio non si riportano i risultati dei best-fit, che sono ovviamente diversi rispetto al caso precedente.

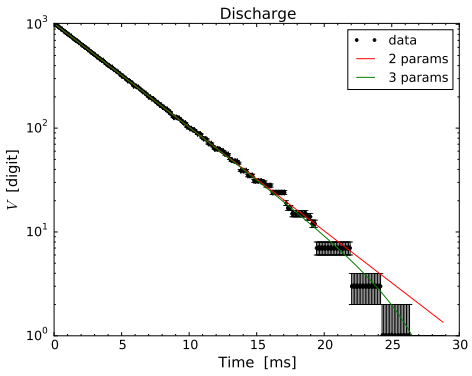

Figura 3. Rappresentazione semilogaritmica dei dati corrispondenti alla fase di scarica nelle condizioni del testo (diverse rispetto a quelle di Fig. 2), assieme alle curve di best-fit a due e tre parametri.

Il grafico è istruttivo per vari motivi: (i) mostra chiaramente gli effetti della digitalizzazione quando il valore digitalizzato è piccolo (e dunque paragonabile al singolo

bit di digitalizzazione); (ii) evidenzia chiaramente gli effetti del piccolo offset incluso nel best-fit a tre parametri, che "piega" la retta (verso il basso di figura, essendo il suo segno negativo); (iii) fa intuire una caratteristica forma "scalettata" nei dati acquisiti, sulla cui origine torneremo nella prossima sezione.

Per la realizzazione pratica del grafico in rappresentazione semilogaritmica, osservate che, quando, come in questo caso, possono esserci valori nulli nel campione, occorre *regolare manualmente* la scala dell'asse verticale: infatti lo zero non è rappresentabile in scala logaritmica, e può succedere che Python scelga in automatico una scala con limite inferiore molto basso, cosa che non ha alcun significato fisico (nell'esempio riportato, la scala è stata fatta partire dal valore 1).

C. Ulteriori considerazioni sul funzionamento di Arduino

L'analisi qualitativa delle Figg. 2 e 3 suggerisce come, nel suo funzionamento in questo specifico esperimento, Arduino non obbedisca sempre alle aspettative. Lo studio svolto in una precedente esperienza aveva dimostrato che l'incertezza convenzionale di ± 1 digit attribuita ai valori digitalizzati poteva essere considerata come una buona stima (anzi, una piccola sovrastima) dell'errore effettivamente compiuto. Solo in alcuni casi si era infatti verificato come la deviazione standard sperimentale calcolata su un ampio campione fosse maggiore di questa incertezza convenzionale.

Osservando i grafici dei residui normalizzati di Fig. 2 risulta evidente come, invece, almeno *per alcuni dei dati registrati* l'incertezza attribuita risulti di fatto sottostimata. Infatti alcuni residui normalizzati schizzano molto al di fuori del range atteso e il grafico mostra, in alcuni casi, un andamento peculiare (i residui normalizzati formano una sorta di sequenze di "creste" con un andamento simile a quello esponenziale). D'altra parte i dati stessi, come si vede molto bene nella rappresentazione semilogaritmica di Fig. 3, presentano talvolta un andamento "scalettato".

In effetti le condizioni di operazione del presente esperimento sono ben diverse rispetto a quelle della misura di d.d.p. continue. Qui, infatti, il segnale da digitalizzare cambia con il tempo. Ricordiamo bene, dal modello di base che conosciamo, come il processo di digitalizzazione richieda del tempo per essere completato: sulla base delle analisi svolte in precedenza, abbiamo addirittura individuato tale tempo, che in questa nota abbiamo chiamato Δt_{dig}. È evidente che, affinché il digitalizzatore possa compiere in maniera fedele il suo compito, la d.d.p. in ingresso dovrebbe rimanere costante per tutto questo (breve) intervallo di tempo. Normalmente, a questo scopo provvede un circuito ulteriore, detto *sample and hold*, che è posto all'ingresso del digitalizzatore e che si occupa di mantenere costante il segnale durante l'intervallo Δt_{dig}. Un modello semplificato per questo circuito è costituito da un condensatore, che viene mantenuto carico (a un valore proporzionale alla d.d.p. che deve essere digitalizzata) per tutto il tempo necessario alla digitalizzazione, e quindi scaricato istantaneamente e completamente per essere caricato a un valore diverso, corrispondente alla nuova d.d.p. da misurare, e quindi permettere una nuova digitalizzazione.

I risultati sperimentali di questa esperienza suggeriscono che questo meccanismo non funziona sempre in modo affidabile. In particolare, sembra che il digitalizzatore modifichi la sua lettura con una sorta di "inerzia", che dà luogo ad alcune delle scalettature osservate nei grafici. È interessante notare che questo comportamento erroneo risulta particolarmente evidente o per bassi valori digitalizzati, dove la sensibilità finita gioca un ruolo importante (l'incertezza relativa del valore digitalizzato è qui maggiore che per grandi valori digitalizzati), oppure in corrispondenza di valori specifici, spesso distribuiti attorno a multipli di 2 (ad esempio attorno a 256, 512, 768, quest'ultimo in maniera molto rilevante).

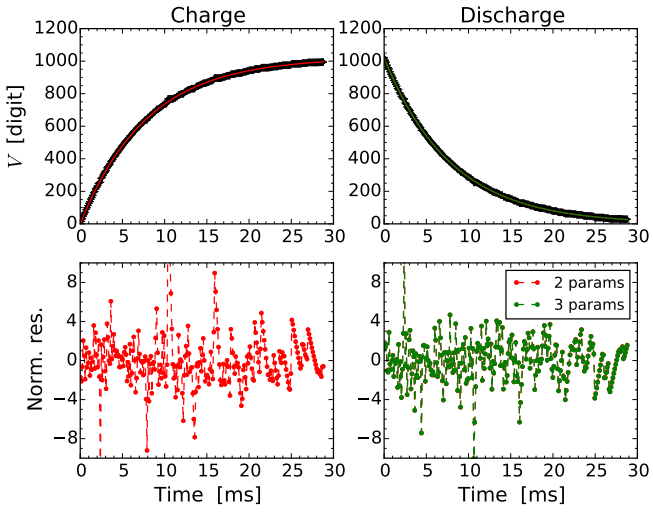

Figura 4. Analogo di Fig. 2, ma con dati ottenuti da medie su 32 cicli di acquisizione.

Il comportamento che stiamo ipotizzando costituisce evidentemente una caratteristica *sistematica* della specifica scheda Arduino in uso. Il carattere sistematico può essere messo in evidenza eseguendo tanti cicli di acquisizione, cioè costruendo un campione di dati, e calcolando media e deviazione standard sperimentale su questo campione. La Fig. 4 mostra un esempio di dati ottenuti sulla base di 32 cicli di acquisizione: il valore graficato è la media e la barra di errore è la maggiore tra errore convenzionale e deviazione standard sperimentale.

Confrontando con Fig. 2 si vede come il risultato sia qualitativamente simile: anche in questo caso i grafici dei residui normalizzati mostrano spikes e andamenti peculiari. Le uniche differenze rilevanti nei risultati dei best-fit sono nel valore del χ^2, che si riduce di circa un centi-

naio (per motivi di spazio si rinuncia all'analisi completa dei risultati).

Se spikes e andamenti peculiari avessero avuto origine statistica, i loro effetti sarebbero diminuiti mediando su diversi cicli di acquisizione. Il risultato ottenuto suggerisce quindi che i "problemi" di Arduino qui messi in evidenza [4] hanno un carattere prevalentemente sistematico. Per attenuarne gli effetti può essere giustificato, in occasioni selezionate e ben motivate, operare come descritto in Appendice.

APPENDICE: OUTLIERS

Questa Appendice discute una procedura di manipolazione dati molto pesante, che tuttavia può essere applicata in determinati e ben motivati casi. Naturalmente occorre sempre dichiararne l'uso con la massima chiarezza e, trattandosi di una procedura sostanzialmente *arbitraria*, è anche necessario cercare delle valide giustificazioni.

La procedura prevede di individuare e isolare quei dati che si discostano in maniera "significativa" dalle previsioni del best-fit. Questi dati prendono il nome di *outliers* e, una volta individuati, diventa possibile ripetere un nuovo best-fit *escludendoli* dal best-fit stesso. In sostanza, nella procedura:

1. si esegue un primo best-fit su *tutti* i dati disponibili;
2. si identificano gli outliers come quei dati che si discostano in valore assoluto dal best-fit per più di una certa soglia;
3. si sovrappone il grafico degli outliers a quello dei dati sperimentali, facendo in modo che gli uni siano ben distinguibili dagli altri;
4. si esegue un nuovo best-fit sul set di dati senza gli outliers;
5. eventualmente si itera la procedura.

I cardini della giustificazione potrebbero essere, per la situazione qui considerata, i seguenti:

1. siamo ragionevolmente convinti che il modello e la funzione che ne deriva siano in grado di descrivere le osservazioni sperimentali;
2. siamo ragionevolmente convinti che gli outliers sono dovuti a un comportamento erroneo, e sostanzialmente sistematico, dello strumento di misura;
3. il best-fit ci serve non per verificare l'affidabilità di un modello, che abbiamo già data per scontata, ma per ottenere la misura indiretta di una qualche quantità fisica (la capacità C del condensatore, nel nostro caso).

La scelta della soglia che determina gli outliers è ovviamente critica e, almeno in linea di principio, il suo effetto andrebbe studiato con attenzione, ripetendo la procedura per diverse scelte della soglia. Qui, naturalmente, ci limiteremo a compiere un solo tentativo e, per esigenze di spazio, considereremo solo uno dei best-fit realizzati in Fig. 2, in particolare quello relativo alla fase di scarica modellata a due parametri, già usato per determinare quantitativamente la capacità del condensatore C.

La Fig. 5 mostra il risultato della procedura: nel pannello superiore sono riportati il grafico dei dati sperimentali (tutti) assieme a quello degli outliers, indicati da tondini di colore blu, e al best-fit a due parametri ottenuto alla fine, cioè escludendo gli outliers. In questo esempio è stata impiegata la rappresentazione semilogaritmica per esigenze di chiarezza tipografica. Inoltre la soglia per l'identificazione degli outliers è stata posta pari a 2 volte l'errore relativo, cioè sono stati definiti come outliers quei punti sperimentali V_j tali che $|V_j - y_j| \geq 2\sigma_j$, con y_j previsione del best-fit (eseguita senza tenere conto della covarianza tra i parametri) e σ_j la somma in quadratura dell'incertezza δy_j e dell'errore equivalente ottenuto propagando l'incertezza δt_j.

Dando confidenza alla natura prevalentemente statistica delle incertezze qui considerate, che abbiamo *arbitrariamente* supposto, si potrebbe in questo caso impiegare a ragion veduta un'affermazione spesso abusata, cioè affermare che gli outliers sono quei dati sperimentali che *distano per oltre 2σ* dalla previsione. Notate che la scelta della rappresentazione con tondini rende evidente quali dati siano stati scartati nel best-fit, in modo da soddisfare l'esigenza di *dichiarare* gli outliers. Il pannello inferiore riporta il grafico dei residui normalizzati ottenuto considerando tutti i dati (markers e linea tratteggiata rossa) oppure escludendo gli outliers (markers e linea tratteggiata blu).

Tabella III. Risultati del best-fit mostrato in Fig. 5 e ottenuto escludendo gli outliers secondo quanto discusso nel testo. Per il best-fit è stata usata l'opzione `absolute_sigma = True` e il numero di gradi di libertà (ndof) è pari a 172 (82 dati sono stati considerati outliers). I dati di partenza sono gli stessi graficati nel pannello "discharge" di Fig. 2, e i risultati di questa tabella possono essere confrontati con i corrispondenti risultati riassunti in Tabb. I, II (il modello è quello a due parametri).

χ^2	a [digits]	τ_S [ms]	cov_{a,τ_S}
205	1021.2 ± 0.3	7.888 ± 0.003	-0.71

In questo esempio, e per la citata scelta della soglia, si individuano 82 outliers su 256 dati, cioè circa 1/3 delle misure viene scartato nel best-fit (valore un po' troppo grande, ma qui mantenuto per esigenze "didattiche"). La Tab. III riporta i risultati del best-fit a due parametri eseguito escludendo gli outliers. I parametri ottenuti risultano compatibili (al limite delle incertezze) con quelli

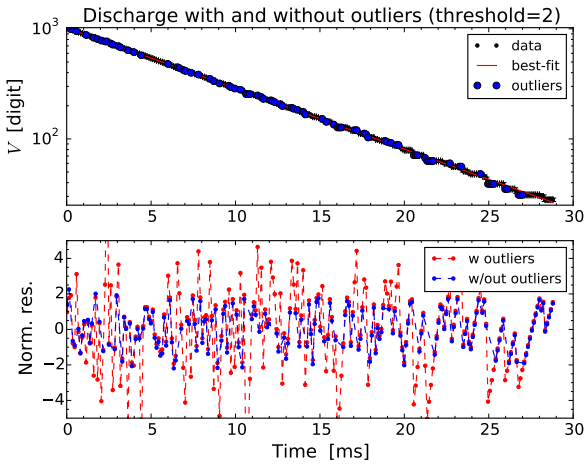

Figura 5. Rappresentazione semilogaritmica dei dati corrispondenti alla fase di scarica nelle stesse condizioni di Fig. 2, assieme ai dati identificati come outliers, qui indicati con tondini blu, e alla curva di best-fit a due parametri eseguita escludendo gli outliers. Il pannello inferiore riporta il grafico dei residui normalizzati considerando tutti i dati (markers e linea tratteggiata rossa) o escludendo gli outliers (markers e linea tratteggiata blu).

riportati in Tab. I, per cui si può concludere che, nel caso specifico qui trattato, i "problemi" di Arduino non hanno un effetto rilevante nella determinazione dei parametri di nostro interesse, in particolare di τ_S. Infatti, anche se gli outliers sono numerosi, essi non sono distribuiti in modo da produrre variazioni rilevanti nella procedura di minimizzazione. Invece, come è ovvio, il χ^2 si riduce sensibilmente. Si ottiene infatti $\chi^2_{rid} = 1.2$, dal che, sempre confidando nel carattere statistico delle incertezze e in quello sistematico degli outliers, potremmo dedurre un *livello di confidenza* superiore al 90 %.

Come già affermato, l'individuazione e rimozione degli outliers va eseguita solo in circostanze specifiche, quando sono ragionevolmente soddisfatte le condizioni elencate sopra. In questa esperienza, relativa alla carica e scarica del condensatore, la procedura non permette di guadagnare significativamente in termini di accuratezza sulla determinazione dei parametri, né sotto altri punti di vista. Tuttavia la sua applicazione, da compiere con calma (a casa), è sicuramente consigliata, soprattutto perché la messa a punto dell'algoritmo necessario è tutt'altro che banale e costituisce un buon esercizio di programmazione, che vale la pena di svolgere in previsione di ulteriori esperienze e necessità.

---

[1] Il datasheet del microcontroller di Arduino dà un'indicazione sulla massima corrente che può essere richiesta alle porte digitali: essa vale nominalmente 40 mA. Nulla si conosce a proposito della resistenza interna, e d'altra parte il concetto di resistenza interna "à la Thévenin" non si applica in modo completo a dispositivi elettronici complicati, come il microcontroller di Arduino. Come regola di massima, è opportuno evitare una richiesta di corrente superiore a pochi mA se si vuole considerare ragionevolmente ideale il generatore di d.d.p. costituito dalle porte digitali di Arduino. Vostri colleghi dello scorso anno (Gianfranco/Maurizio/Silvio, thanks!) hanno comunque eseguito una misura stile Thévenin, ottenendo una resistenza interna prossima a 20 ohm.

[2] Ricordate sempre che i risultati delle misure dipendono strettamente dalle effettive condizioni operative e dalla specifica scheda Arduino impiegata. Negli esempi qui considerati Arduino è stato collegato a un computer portatile alimentato a batteria e scollegato dalla linea di terra. Questo potrebbe comportare una situazione particolarmente favorevole in termini di immunità da disturbi ("rumore").

[3] L'incertezza sui parametri di best-fit è generalmente minore per i fit a due parametri. Questo è atteso visto che nella previsione andrebbe considerata la covarianza, e che essa è negativa per alcuni parametri. Osservate che, a causa dell'uso (*arbitrario*) dell'opzione `absolute_sigma = True`, l'incertezza sui parametri è quella *standard* che, nei casi in cui $\chi^2_{rid} > 1$, risulta minore rispetto all'incertezza *asintotica*. Infatti l'incertezza asintotica è $\sqrt{\chi^2_{rid}}$ volte quella standard. Negli esempi considerati questo fattore è sufficientemente piccolo (sicuramente minore di un ordine di grandezza) da non suscitare particolari preoccupazioni per la scelta dell'opzione.

[4] Il problema è in realtà tipico per tutti i convertitori analogico/digitali, non solo per Arduino. Talvolta la figura di merito che lo caratterizza può essere la *linearità differenziale*, cioè la misura della linearità in presenza di differenze (in questo caso, da intendersi in funzione del tempo) di segnali in ingresso.