Carica/scarica condensatore con Arduino

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

(Dated: version 6 - Lara Palla e Francesco Fuso, 10 novembre 2015)

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. 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. Dunque 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. Esistono delle limitazioni, legate fondamentalmente alla relativa lentezza di operazione di Arduino, cioè al suo rate di campionamento limitato (dell'ordine della decina di kSa/s), che impediscono la registrazione di fenomeni che avvengano con un tempo caratteristico troppo breve: come chiariremo nel seguito, nella configurazione impiegata i risultati migliori si ottengono per $\tau \sim 5 - 50$ ms, ottenibile con opportune scelte dei valori di R e C.

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 nel tempo e di registrarne il valore grazie alla presenza di convertitori analogico/digitali collegate alle porte di ingresso, per esempio quella collegata al pin `A0` già impiegata per la misura di una tensione continua. 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, `GND` (boccola nera) indica la connessione di massa, o terra, che è ovviamente necessario effettuare 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, si spegne la porta digitale, e si ricomincia l'acquisizione temporizzata per seguire la fase di scarica. Al termine del processo saranno 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 y_j della d.d.p., in unità arbitrarie di digitalizzazione,

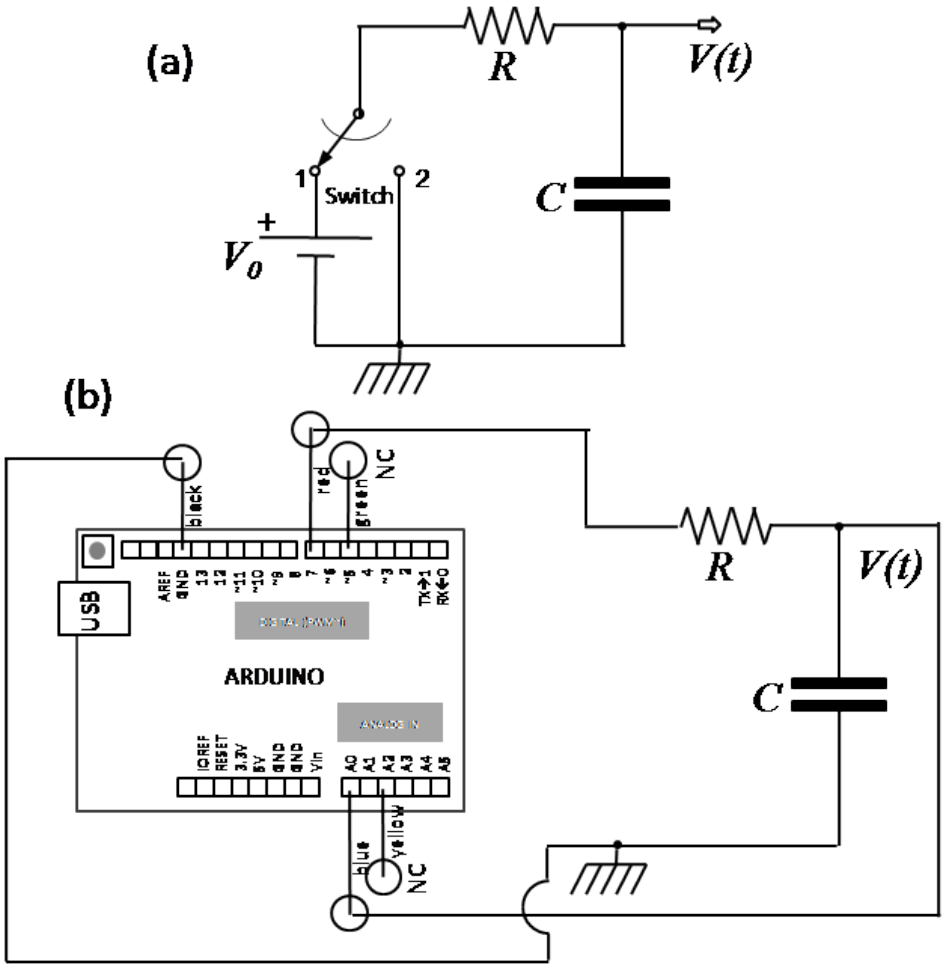

Figura 1. Schema concettuale dell'esperienza (a) e sua realizzazione con Arduino (b). Nel pannello (b) è rappresentata una visione molto schematica e non in scala della scheda Arduino Uno rev. 3 SMD edition usata nell'esperienza. Ci sono cinque collegamenti a altrettanti pin della scheda che terminano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

o *digit*s. 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 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, SCRIPT E SKETCH

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 il rate di campionamento, ovvero il ritardo Δt tra due misure successive, 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.

Chiariamo subito un aspetto caratteristico dell'esperienza: in essa vogliamo "spingere al massimo" il rate di campionamento di Arduino in modo da registrare fenomeni sufficientemente rapidi, con una *risoluzione temporale* inferiore al millisecondo. Dal punto di vista concettuale, questa richiesta potrebbe essere non necessaria, dato che sarebbe sufficiente scegliere una costante tempo abbastanza grande da poter operare lentamente. Tuttavia uno scopo addizionale di questa esperienza è proprio quello di verificare le potenzialità di Arduino per il campionamento di transienti ("abbastanza") rapidi, mettendo in luce limiti e problemi connessi con la velocità di campionamento.

Allo scopo di consentire un'acquisizione significativa, l'esperienza richiede di determinare a priori l'intervallo temporale di campionamento Δt, che va espresso in unità di 100 μs all'interno dello script di Python. La scelta di Δt dipende dalla costante tempo, a sua volta determinata dai valori di resistenza e capacità in uso nell'esperimento. Come è noto, nella fase di carica il tempo caratteristico τ rappresenta l'intervallo temporale nel quale la carica aumenta fino al valore $(e-1)/e$ (corrispondente grosso modo a $2/3$) del valore asintotico, mentre nella fase di scarica il tempo caratteristico τ rappresenta l'intervallo temporale in cui la carica diminuisce fino ad arrivare alla frazione $1/e$ del valore massimo (iniziale). Il valore asintotico, o lo zero, della d.d.p. ai capi del condensatore, a seconda della fase esaminata, viene idealmente raggiunto dopo un tempo molto lungo, tendente a infinito. Dal punto di vista pratico, la carica/scarica di un condensatore è significativamente completata dopo un intervallo di tempo pari a $(4-8)\tau$. Dunque nella vostra acquisizione, composta di 250 misurate intervallate *nominalmente* di Δt, e dunque tali da coprire l'intervallo temporale complessivo $t_{tot} \sim 250\Delta t$, conviene scegliere Δt in modo tale che si abbia $t_{tot} \simeq (4-8)\tau$ se si vuole essere certi di raggiungere in maniera ragionevolmente soddisfacente la condizioni asintotica di carica o scarica. D'altra parte, tenendo conto del fatto che poi i dati raccolti devono essere impiegati in un best-fit, e che questo best-fit deve servire (anche) a determinare la costante tempo caratteristica τ, è bene limitare l'intervallo di tempo t_{tot} a non più di 3 τ. Infatti la valutazione del tempo caratteristico è ovviamente molto poco sensibile quando si considerano i valori prossimo all'asintoto.

A. Lo script di Python

Lo script di Python che controlla Arduino è piuttosto semplice, e abbastanza simile a quello già impiegato nell'esperienza della misura di tensioni continue. A differenza di quella esperienza, qui non è richiesto eseguire un ciclo di acquisizioni, dunque non compare alcun ciclo dedicato a questo scopo.

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 250 righe e due colonne (tempo t_j in microsecondi, valore della d.d.p. digitalizzata y_j in digit, con j che corre da 0 a 249), 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 250 coppie di dati per la fase di carica, e quindi, dopo averne eseguito la misura, le 250 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 `caricascarica_v1.py`, è riportato con qualche commento qui nel seguito.

```
import serial # libreria per gestione porta seriale (USB)
import time # libreria per temporizzazione
Directory='../dati_arduino/'   # nome directory dati
                                          # << DA CAMBIARE SECONDO NECESSITA'
FileName='pippo'  # parte comune nome file << DA CAMBIARE SECONDO NECESSITA'
```

```
FileNameC=(Directory+FileName+'_C.txt') # crea nome file dati carica
FileNameS=(Directory+FileName+'_S.txt') # crea nome file dati scarica
ard=serial.Serial('/dev/ttyACM0',9600)  # apre porta seriale (occhio alla sintassi, dipende
                              # dal sistema operativo!)
time.sleep(2)   # aspetta due secondi per evitare casini

ard.write(b'5') # scrive il carattere per l'intervallo di campionamento
                # in unità di 100 us << DA CAMBIARE A SECONDA DEI GUSTI
                # l'istruzione b indica che è un byte (carattere ASCII)

time.sleep(2) # aspetta due secondi per evitare casini

outputFileC = open(FileNameC, "w" ) # apre file dati carica per scrittura
outputFileS = open(FileNameS, "w" ) # apre file dati scarica per scrittura

# loop lettura dati da seriale (500, di cui 250 per carica e 250 per scarica)
for i in range (0,500):
    data = ard.readline().decode() # legge il dato e lo decodifica
    if data:
        if i<250:
            outputFileC.write(data) # scrive i primi 200 in file di carica
        else:
            outputFileS.write(data) # scrive i secondi 200 in file di scarica
outputFileC.close() # chiude il file dei dati di carica
outputFileS.close() # chiude il file dei dati di scarica
ard.close() # chiude la comunicazione seriale con Arduino
print('end') # scrive sulla console che ha finito
```

B. Lo sketch di Arduino

Anche lo sketch di Arduino ha forma e struttura 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. Richiamiamo brevemente qui di seguito le istruzioni più significative di ogni blocco.

Nel blocco di dichiarazione compare l'istruzione `long t[250];` che serve a definire l'array `t[i]` che contiene gli istanti t_j a cui avvengono le varie misure. Questi istanti sono di fatto "misurati" e torneremo in seguito sugli eventuali problemi connessi alla "misura" dei tempi. Per il momento accontentiamoci di constatare che questo array deve essere definito come `long`, cioè un intero composto da 4 bytes. Esso, infatti, contiene il ritardo in microsecondi a partire dall'istante "iniziale", quello in cui la variabile viene azzerata (nominalmente all'inizio della fase di carica e all'inizio di quella di scarica). Poiché l'esperienza prevede di eseguire 250 misure intervallate, nominalmente, fino a 900 μs l'una dall'altra, è evidente che il *numero intero* di microsecondi da scrivere nell'array può superare il valore massimo compatibile con una variabile di tipo `int`. Pertanto la dichiarazione deve essere per un array di variabili `long`, nonostante questa scelta comporti una maggiore occupazione della memoria del microcontroller. Inoltre nello stesso blocco compare l'istruzione `long StartTime;`: questa variabile viene letta all'inizio delle due fasi e, attraverso un'opportuna operazione di differenza, consente in pratica di "azzerare" il cronometro all'inizio della carica e della scarica.

Nel blocco di inizializzazione compaiono le due righe di programma `bitClear(ADCSRA,ADPS0);` e `bitClear(ADCSRA,ADPS2);`. Queste istruzioni agiscono direttamente sui "registri" del microcontroller, determinando sostanzialmente la rapidità con cui vengono compiute le operazioni di digitalizzazione. Esse, infatti, permettono di aumentare l'accuratezza nel timing, in modo che essa scenda sotto il millisecondo, come da obiettivo della nostra esperienza. L'impostazione può avere un "costo" in termini di accuratezza di digitalizzazione [2].

Infine il loop delle misure ha una sintassi piuttosto auto-esplicativa. Notate come anche in questo caso la prima lettura di ogni fase venga "cestinata" (messa nella variabile trash che poi non viene considerata) e come la variabile `StarTime` venga rinfrescata all'inizio di ogni ciclo, in modo da avere misure di t_j, cioè valori dell'array `t[i]` che rappresentano nominalmente i ritardi rispetto all'inizio delle due fasi. Osservate anche 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.

Lo sketch, che è disponibile nei computer di laboratorio

e anche in rete, sotto il nome di `carich.ino`, è riportato nel seguito con qualche commento.

```
// Blocco dichiarazioni
const int analogPin_uno=0; // Definisice la porta A0 usata per la lettura
const int digitalPin_uno=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 V1[250]; // Definisce l'array intero V1
long t[250]; // 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)
int trash; // Definisce la variabile di trash (prima lettura, da cestinare)

// Istruzioni di inizializzazione
void setup()
  {
   Serial.begin(9600); // Inizializza la porta seriale a 9600 baud
   Serial.flush(); // Pulisce il buffer della porta seriale
   pinMode(digitalPin_uno,OUTPUT); // Definisce digitalPin_uno come output
   digitalWrite(digitalPin_uno,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 è 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_uno,HIGH); // Pone digitalPin_uno a livello alto per la carica
    StartTime=micros(); // Misura il tempo iniziale con l'orologio interno (lettura in us)
    trash = analogRead(analogPin_uno); // Mette la prima lettura, da cestinare, in trash
    for(i=0;i<250;i++) // Loop per la carica
      {
          t[i]=micros()-StartTime; // Legge il timestamp in us, sottrae lo StartTime e mette il risultato i
          V1[i]=analogRead(analogPin_uno); // Legge analogPin e lo mette in array V1
          delayMicroseconds(delays); // Aspetta tot us
      }

    for(i=0;i<250;i++) // Loop per la scrittura su porta seriale dei 250 dati della carica
      {
        Serial.print(t[i]); // Scrive t[i]
        Serial.print(" "); // Mette uno spazio
        Serial.println(V1[i]); // Scrive V1[i] e va a capo
      }

   delay(1000); // Aspetta 1000 ms per completare la carica
   digitalWrite(digitalPin_uno,LOW); // Pone chargePin a livello basso per la scarica
```

```
    trash = analogRead(analogPin_uno); // Mette la prima lettura, da cestinare, in trash
    for(i=0;i<250;i++) // Loop per la carica
      {
          t[i]=micros()-StartTime; // Legge il timestamp in us, sottrae lo StartTime e mette il risultato i
          V1[i]=analogRead(analogPin_uno); // Legge analogPin e lo mette in array V1
          delayMicroseconds(delays); // Aspetta tot us
      }

    for(i=0;i<250;i++) // Loop per la scrittura su porta seriale dei 250 dati della carica
      {
        Serial.print(t[i]); // Scrive t[i]
        Serial.print(" "); // Mette uno spazio
        Serial.println(V1[i]); // Scrive V1[i] e va a capo
      }
  start=0; // Annulla il flag
    Serial.flush(); // Pulsice il buffer della porta seriale (si sa mai)
}
```

IV. CONSIDERAZIONI SULLA MISURA E DETERMINAZIONE DELLE INCERTEZZE

Questa esperienza illustra bene come la corretta attribuzione delle incertezze di misura sia un'operazione tutt'altro che banale anche in configurazioni semplici dal punto di vista concettuale e pratico.

A. Misura di y_j

Innanzitutto, notiamo che, se il nostro obiettivo è quello di valutare i tempi caratteristici di carica e scarica attraverso best-fit dei dati acquisiti, *non c'è alcun bisogno* di esprimere la d.d.p. ai capi del condensatore in unità fisiche. Infatti, detto ξ il fattore di calibrazione, e supponendo per il momento lineare il legame tra valore digitalizzato e grandezza fisica campionata, nelle equazioni di nostro interesse esso compare a moltiplicare, o dividere, gli esponenziali. In altre parole, e con ovvio significato dei termini, $V(t) = \xi y(t)$, da cui, limitandoci a scrivere la sola equazione per la fase di scarica, $y(t) = (V_0/\xi)\exp(-t/\tau_S)$. Dunque il fattore di calibrazione non influenza la valutazione di τ_S, dato che esso compare nel coefficiente che moltiplica l'esponenziale, il quale sarà lasciato libero come parametro nel best-fit. Di conseguenza, siamo (inizialmente) autorizzati a porre l'incertezza, qui indicata con δy_j, pari a 1 digit, cioè a considerare il solo errore di digitalizzazione di origine stocastica, trascurando completamente quello di calibrazione. Infatti, come abbiamo dimostrato con l'esperienza precedente, l'incertezza ± 1 digit costituisce una ragionevole (e piccola) sovrastima della deviazione standard sperimentale σ_y misurata su un campione.

Come vedremo nel seguito, questa scelta, pur essendo piuttosto ben motivata, comporta alcuni problemi. Qui ci limitiamo ad anticiparne alcuni:

1. non è affatto detto che la deviazione standard sperimentale rimanga al di sotto dell'unità quando il digitalizzatore è operato a elevato rate di campionamento *per la misura di segnali transienti*, come qui ci accingiamo a fare;

2. la condizione di linearità tra valore digitalizzato e d.d.p. in ingresso, che è alla base dell'espressione scritta sopra, potrebbe non essere rispettata. Infatti l'esperienza che abbiamo già compiuto su Arduino mostra, in accordo con i datasheets, che la relazione tra d.d.p. in ingresso e suo valore digitalizzato è ben descritta dall'equazione di una retta che *non* passa per l'origine, cioè la relazione è lineare ma c'è un *offset* da considerare. Per tenerne conto, potremo aggiungere un termine costante alla funzione di fit (quando rilevante). In accordo con i datasheets e con i risultati della precedente esperienza, questo termine costante dovrebbe essere negativo e piccolo (pochi digit). In alternativa, potremmo limitarci a considerare misure in cui il valore della d.d.p. si mantiene significativamente diverso da zero, cioè quelle per le quali l'offset è atteso contare poco.

B. Misura di t_j

È relativamente semplice, per un dispositivo elettronico come Arduino, ottenere un'elevata accuratezza nella misura dei tempi. I tempi di Arduino sono scanditi misurando i periodi di oscillazione di un oscillatore (al quarzo) contenuto nella scheda, definito *clock*. La frequenza nominale di oscillazione è 16.000 MHz, per cui un ciclo di oscillazione ha un periodo nominale $T = 62.5$ ns. L'incertezza (stocastica) nel conteggio dei cicli, e quindi nella misura dei tempi, ha idealmente un limite minimo di tale ordine di grandezza [3]. Comparata con gli intervalli

Δt della nostra esperienza, questo limite nell'incertezza produce sicuramente effetti trascurabili.

D'altra parte, l'operazione di Arduino è gestita da un programma che gira nel microcontroller della scheda. Come sappiamo, il microcontroller esegue un gran numero di operazioni, non tutte finalizzate all'esecuzione del programma. Questo fa sì che possano comparire in maniera non deterministica degli stati di attesa, o latenze, che rendono poco controllata, o poco ripetibile, la misura dei tempi così come congegnata in questa esperienza. Inoltre è evidente dallo sketch che la misura dei tempi non avviene in contemporanea con la digitalizzazione della d.d.p. presente alla porta analogica: infatti le due misure corrispondono a due linee distinte di programma (sono due operazioni distinte), che vengono eseguite sequenzialmente. Una situazione analoga si ha anche per la determinazione della variabile `StartTime` usata per "azzerare" il cronometro nella fase di carica e di scarica, che avviene anch'essa con un'istruzione di programma distinta da quella di controllo della porta digitale.

Tutto questo comporta una difficoltà di principio nel determinare l'incertezza δt_j. Nel seguito presenteremo un tentativo di stima sperimentale; per il momento limitiamoci a proporre due possibilità, che hanno carattere diametralmente opposto, pur essendo entrambe motivate da considerazioni fisiche. Infatti possiamo a nostra scelta

- dichiarare trascurabile l'incertezza sulla misura dei tempi rispetto a quella sulla digitalizzazione, per esempio ponendola pari a $\pm$ 1 μs (il microsecondo è l'unità di misura dei tempi nella presente esperienza) [4];

- sovrastimare *arbitrariamente* tale incertezza individuandola come metà dell'intervallo di campionamento Δt: infatti possiamo essere sicuri che la misura dei tempi avviene *tra* due campionamenti successivi della d.d.p..

Salvo ulteriori considerazioni che eventualmente presenteremo nel seguito, la prima delle due scelte è probabilmente più corretta.

V. ESEMPI DI MISURE

Per l'esempio qui considerato è stato scelto $R = (7.10 \pm 0.06)$ kohm e $C = 0.47$ μF (nominale, con tolleranza 10 %). La resistenza impiegata limita la massima corrente richiesta alla porta digitale di Arduino a $V_0/R \leq 1$ mA, che dovrebbe rendere trascurabile il ruolo della resistenza interna del generatore (porta digitale di Arduino). Il tempo caratteristico previsto è $\tau_{att} \simeq 3.3$ ms. Avendo a disposizione 250 punti di misura, è stato scelto l'intervallo di campionamento nominale $\Delta t = 100$ μs, che comporta $t_{tot,att} = 250 \times 0.1 = 25$ ms, corrispondente a oltre 7τ.

La Fig. 2 mostra i dati (valore digitalizzato y_j in funzione del tempo t_j, convertito in ms per una migliore leggibilità delle scale) in rappresentazione ordinaria; le incertezze sono $\delta y_j = \pm 1$ digit e $\delta t_j = \pm 1$ μs. Si osserva subito un risultato inatteso: i dati sono registrati in un intervallo temporale complessivo, $t_{tot} > t_{tot,att}$. Come esamineremo meglio in Sez. VI, questo è dovuto sostanzialmente alla presenza di ritardi nell'esecuzione dei cicli di programma.

L'analisi di questi dati è stata compiuta secondo funzioni modello che descrivono gli andamenti della carica e scarica del condensatore nel tempo. Le funzioni sono

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

La presenza del terzo parametro, denominato c, tiene conto dell'eventuale offset del digitalizzatore; notate che per la fase di carica l'aggiunta di questo parametro non ha senso, risultando esso pressoché completamente (anti)correlato con il parametro a. Nel best-fit non si è tenuto conto dell'incertezza δt_j tramite propagazione dell'errore; si è comunque fatta una prova usando la propagazione dell'errore, che ha dimostrato effetti trascurabili (la riduzione del χ^2 è dell'ordine di qualche punto percentuale) [5]. Le curve di best-fit ottenute sono sovrapposte ai dati sperimentali: visivamente non si nota alcuna differenza tra le due funzioni modello (sia per la carica che per la scarica) e l'accordo con i dati risulta piuttosto buono.

Tuttavia il grafico dei residui normalizzati indica discrepanze elevate (notate la scala verticale dei grafici) e soprattutto degli andamenti particolari, che si svolgono in maniera piuttosto simile in tutte le misure. In particolare si distinguono dei picchi (positivi o negativi) corrispondenti a dei punti singoli, cioè a dei singoli campionamenti, che potrebbero essere dovuti a *spikes* nel segnale in ingresso alla porta analogica (inclusi fenomeni casuali dovuti ad accoppiamento con sorgenti di rumore). Oltre a questi singoli punti, che danno un forte contributo al χ^2, si osserva la ripetizione di andamenti simili a esponenziali decrescenti o crescenti (nella fase di carica e di scarica). Essi sono dovuti a un comportamento caratteristico di Arduino nella digitalizzazione di segnali transienti: come vedremo meglio nel seguito, l'andamento dei valori digitalizzati non è "continuo", ma presenta delle "scalettature". In altre parole, è come se il valore digitalizzato rimanesse costante per un certo intervallo di tempo, per poi cambiare bruscamente. Poiché la funzione modello usata nel fit contiene degli esponenziali, è chiaro che i residui abbiano anche un andamento esponenziale. Torneremo in Sez. VI su questo fastidioso aspetto.

I risultati dei best-fit sono riassunti in Tab. I e Tab. II (il numero di gradi di libertà è 248 o 247 per i fit a due o tre parametri).

Senza entrare nello specifico dei diversi valori ottenuti come risultato dei best-fit, 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;

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.

Carica				Scarica			
χ^2	a [digits]	τ_C [ms]	c [digits]	χ^2	a [digits]	τ_S [ms]	c [digits]
1400	1023.1 ± 0.2	4.162 ± 0.004	-	1290	1021.6 ± 0.8	4.133 ± 0.004	-
-	-	-	-	1148	1021.6 ± 0.7	4.153 ± 0.005	-1.2 ± 0.2

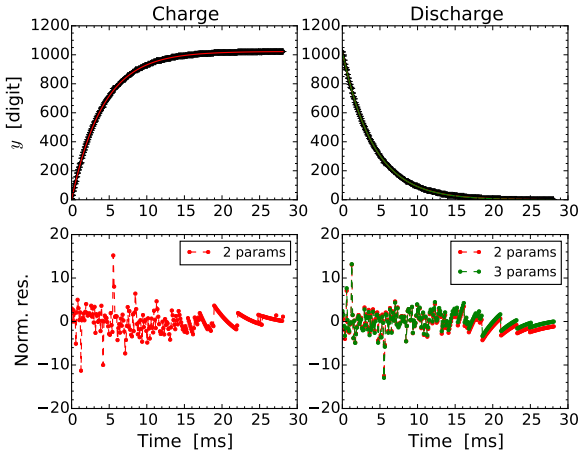

Figura 2. Esempio di misura (carica e scarica, rispettivamente a sinistra e destra di figura) eseguita con la scelta dei valori R e C riportati nel testo; i punti, corredati delle barre di errore, rappresentano dati sperimentali, le linee continue le curve ottenute secondo i best-fit descritti nel testo.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

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}	$\mathrm{cov}_{a,c}$	$\mathrm{cov}_{\tau_S,c}$
0.65	-	-	-0.70	-	-
-	-	-	-0.53	-0.21	-0.63

- l'introduzione del terzo parametro nella funzione modello per la scarica non modifica in maniera sostanziale l'esito dei fit, anche se si osserva una (piccola) riduzione del χ^2;
- il segno di c e il suo valore numerico sono in accordo con le aspettative (basate sui datasheets e sulla precedente esperienza);
- i valori degli altri parametri ottenuti dai best-fit suonano generalmente accettabili.

In particolare, il parametro a è ragionevole rispetto alle aspettative (esso è prossimo a 1023, che corrisponde al massimo valore digitalizzato dalla scheda, quello che deve essere asintoticamente raggiunto nella carica/scarica del condensatore) [6]. Inoltre questi 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 con cui la routine di best-fit individua questi parametri è superiore 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 [7], potremmo dedurre da $\tau_S = (4.172 \pm 0.006)$ ms e dalla misura (indipendente) di $R = (7.10 \pm 0.06)$ kohm un valore per la capacità incognita del condensatore $C = (0.588 \pm 0.006)$ μF (per altro, in disaccordo con il valore nominale del condensatore stesso, anche considerando la tolleranza dichiarata), con una accuratezza sostanzialmente limitata da quella sulla misura di R.

A. 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 $1/\tau_S$. La rappresentazione semilogaritmica è usata in Fig. 3 per gli stessi dati di Fig. 2 (solo quelli relativi alla fase di scarica).

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 l'offset negativo); (iii) fa intuire la caratteristica forma

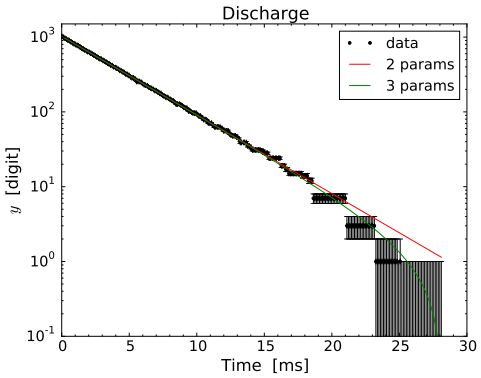

Figura 3. Stessi dati e curve di best-fit per la fase di scarica riportati in Fig. 2, rappresentati qui in carta semilogaritmica.

"scalettata" dei dati acquisiti. Per la realizzazione pratica del grafico in rappresentazione semilogaritmica, osservate che, quando, come in questo caso, ci sono dei 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 0.1 in modo da mostrare dei bei pezzi delle barre di errore).

Notate poi che, nonostante i dati possano essere facilmente linearizzati manipolandoli in modo da considerare la grandezza $y' = \log(y)$, l'esecuzione di un best-fit analitico (del minimo χ^2) a una retta è operazione "delicata". Infatti la presenza di valori nulli in y rende impossibile determinare l'incertezza su y' attraverso propagazione degli errori. Di conseguenza questa strada non verrà qui percorsa.

VI. ULTERIORI MISURE E ANALISI

Quanto presentato finora, in particolare l'analisi dei dati con best-fit numerico a due parametri e la determinazione dei tempi caratteristici di carica e di scarica, può essere considerato un buon obiettivo per l'analisi dell'esperienza compiuta in laboratorio. Resta però un po' di insoddisfazione per la qualità dell'analisi stessa. Questa Sezione riporta alcune ulteriori osservazioni (su misure e analisi) da considerare come "opzionali" (fortemente sconsigliate per l'esecuzione dell'esperienza in laboratorio, magari di possibile interesse per altre attività, per esempio per la relazione semestrale). Per brevità, tali osservazioni sono riferite alla sola analisi del processo di scarica.

A. Analisi del timing

In quanto riportato prima, abbiamo lasciato in dubbio la valutazione dell'accuratezza del timing del campionamento, che è invece un aspetto interessante da analizzare. A questo scopo è stato costruito automaticamente un campione realizzato da 50 acquisizioni, tutte con la stessa scelta di R e C. Delle tante acquisizioni eseguite sono stati considerati i valori di t_j registrati ed è stato creato un unico array di "differenze" $\Delta t_{mis} = t_j - t_{j-1}$. Secondo le attese, queste differenze dovrebbero corrispondere all'intervallo temporale nominale Δt tra due campionamenti consecutivi impostato via script di Python come $\Delta t = 100$ μs, in questo caso). L'analisi statistica, di cui si riporta l'istogramma delle occorrenze in Fig. 4 (pannello superiore), mostra una distribuzione molto peculiare, con pochissimi bin occupati. Il valore medio che si ottiene è $\overline{\Delta t}_{mis} = 112.8$ μs, con deviazione standard sperimentale $\sigma_{\Delta t_{mis}} = 2.1$ μs. Poiché $\overline{\Delta t}_{mis} > \Delta t$, è ovvio che le acquisizioni coprano un intervallo temporale complessivo $t_{tot} > t_{att}$.

Il motivo per cui l'intervallo temporale effettivo risulta maggiore di quello impostato può essere facilmente interpretato come una conseguenza delle modalità di operazione di Arduino in questa esperienza, in cui il timing dei campionamenti è affidato a un programma che gira nel microcontoller. In sostanza, c'è un ritardo tra campionamento e misura dei tempi, che potrebbe corrispondere grossolanamente al tempo che Arduino impiega per eseguire la digitalizzazione. Questo ritardo, dato dalla differenza $\overline{\Delta t}_{mis} - \Delta t = (12.8 \pm 2.1)$ μs, è apparentemente distribuito solo su pochissimi valori discreti. Notate che, di per sé, la presenza di un ritardo non modifica la significatività dei best-fit effettuati. Infatti "tutte" le misure effettuate hanno (all'incirca) lo stesso ritardo, e la sua presenza può dunque essere "inglobata" nella valutazione tramite best-fit del termine costante a. Lo spread sperimentale di questo ritardo, che corrisponde a $\sigma_{\Delta t_{mis}}$ (qui determinato su un campione di 5×10^4 dati), può essere assunto come stima dell'incertezza da attribuire alla misura di t_j nella nostra esperienza. Si vede come la scelta fatta precedentemente ($\delta t_j = 1$ μs) costituisca una (piccola) sottostima del valore ora determinato. In ogni caso la sostanza non cambia, e l'incertezza sulla misura dei tempi può continuare a essere considerata trascurabile.

B. Problemi di digitalizzazione dei transienti

I risultati dell'esperienza dimostrano in modo piuttosto inequivocabile che porre l'incertezza $\delta y_j = 1$ per tutto il set di dati è una *sottostima* dell'errore effettivo che Arduino compie quando deve digitalizzare segnali transienti, almeno nelle condizioni dell'esperimento qui trattato. Questa sottostima crea un'ovvia sovrastima del χ^2 ottenuto dai best-fit.

Il problema di individuare la corretta incertezza δy_j non ha una facile soluzione. Se l'origine del problema fos-

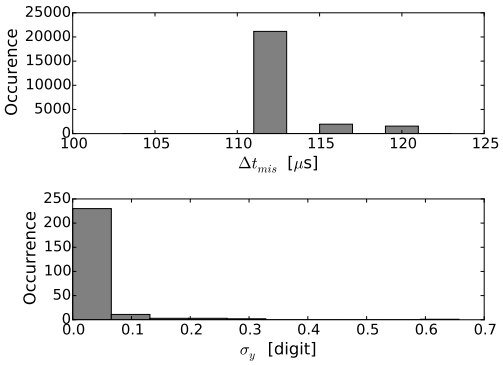

Figura 4. Istogrammi delle occorrenze per la distribuzione degli intervalli temporali (pannello superiore) tra misure successive e per i valori digitalizzati (pannello inferiore), ottenuti come descritto nel testo.

se di natura puramente stocastica, cioè se le discrepanze fra lettura e valore attuale della d.d.p. in ingresso alla porta analogica fossero solo dovute a fluttuazioni (come ad esempio gli spikes dovuti a rumore, che si osservano talvolta), allora ripetendo tante volte la stessa misura si potrebbe determinare la deviazione standard σ_y e usare questa come misura dell'incertezza. La Fig. 4 (pannello inferiore) riporta l'istogramma delle occorrenze di σ_y determinato sul campione di 50 acquisizioni ripetute. Si vede come la distribuzione sia molto stretta e come i valori di σ_y siano sostanzialmente compatibili con quelli ottenuti nella misura di tensioni continue. Infatti l'origine principale delle discrepanze osservate nelle nostre misure ha un'origine almeno parzialmente sistematica, essendo dovuta al particolare funzionamento di Arduino, che si ripete in maniera simile per tutte le acquisizioni [8].

Dovremmo dunque determinare un modo alternativo (e necessariamente *arbitrario*) per aumentare l'incertezza δy_j, in modo da renderla più compatibile con le specifiche caratteristiche della misura. Quella esplorata qui nel seguito è una strada potenzialmente percorribile, anche se anch'essa indicata soprattutto per i casi in cui è presente rumore che evolve su una scala temporale molto più piccola di quella del segnale di interesse.

In queste situazioni un utile strumento di analisi può essere lo *smoothing* dei dati sperimentali. Lo smoothing è una procedura che prevede di manipolare i dati sostituendo a ognuno di essi il risultato di una sorta di *media mobile* su un certo numero di punti (misure) precedenti e successivi. Naturalmente, per non modificare in maniera radicale gli andamenti temporali analizzati, occorre scegliere blocchi "piccoli", cioè fare la media mobile su pochi punti. Nel caso di segnali studiati in funzione del tempo, lo smoothing corrisponde a mediare temporalmente su un breve intervallo, o, se si preferisce, ad applicare un "filtro passa-basso" ai segnali originari se si preferisce usare il linguaggio dell'"analisi spettrale" (che incontreremo in futuro). In questo modo gli andamenti temporali vengono smussati, e le variazione più rapide fortemente attenuate.

Alcuni pacchetti di Python, per esempio `convolve`, riportano numerosi algoritmi di smoothing, nei quali è possibile selezionare il peso che si dà ai vari elementi della media mobile. Per l'esempio presentato in Fig. 5 si è costruita una piccola procedura che calcola la media mobile aritmetica (tutti gli addendi hanno lo stesso peso) su 5 punti (2 precedenti e 2 successivi, di conseguenza nel campione si perdono i 2 primi e i 2 ultimi dati sperimentali). Per tenere conto del problema dell'incertezza, la stessa procedura è stata impiegata per creare delle barre di errore artificiali, che sono state poste pari alla "deviazione standard" del (piccolo) campione usato per la media mobile. I dati considerati qui sono gli stessi di Fig. 3 e quindi il confronto con quella figura illustra in maniera immediata l'esito dell'operazione eseguita. Nella prima parte del decadimento, in cui sono presenti variazioni rapide del valore dei conteggi, la nuova determinazione dell'incertezza produce un sensibile aumento delle barre di errore, "appiattendo" i residui normalizzati. L'effetto diminuisce a tempi lunghi, cioè sulla coda dell'esponenziale. A questi tempi è tuttavia visibile l'ammorbidimento delle variazioni dovuto allo smoothing. Il best-fit a tre parametri condotto sui dati smoothed conduce a parametri τ_S e c in accordo grossolano (entro 5 barre di errore) con quelli di Tab. I [9] e, ovviamente, a una sensibile riduzione del χ^2_{rid}, che assume un valore prossimo all'unità.

Se siete interessati, potete cercare una vostra strada di trattamento dati (anche usando un campione di numerose acquisizioni) per cercare di ridurre i problemi che Arduino mostra nell'acquisizione di transienti.

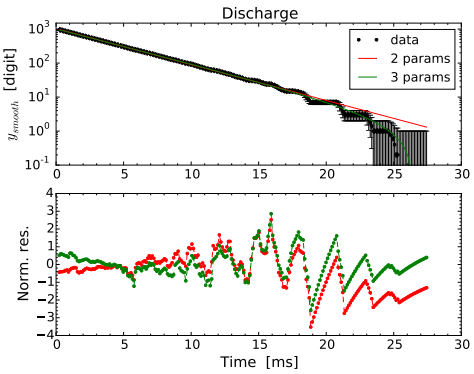

Figura 5. Analogo di Fig. 3, ma con manipolazione dei dati tramite smoothing (media mobile su 5 punti) e determinazione arbitraria dell'incertezza secondo quanto descritto nel testo.

[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 Thevenin" 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. Qualcuno di voi (Gianfranco/Maurizio/Silvio, thanks!) ha comunque eseguito una misura stile Thevenin, ottenendo una resistenza interna prossima a 24 ohm

[2] L'operazione compiuta assomiglia, per certi versi, all'*overclocking* che alcuni smanettoni fanno per le CPU dei propri computer.

[3] Nella stima ("ideale") presentata si suppone che la misura dei tempi sia legata direttamente alla frequenza del clock, mentre in realtà essa avviene contando multipli interi (generalmente potenze di 2) del periodo di oscillazione. Inoltre si trascurano eventuali cause di incertezza legate a fluttuazioni delle condizioni di operazione, per esempio variazioni di temperatura, che potrebbero modificare la frequenza di clock.

[4] Che l'incertezza δt_j produca effetti trascurabili ai fini del best-fit va, in linea di principio, dimostrato usando la propagazione dell'errore. La propagazione dell'errore stabilisce che il contributo all'incertezza sulla misura della d.d.p. ai capi del condensatore dovuta all'incertezza sulla misura dei tempi è proporzionale a $y_j \delta t_j / \tau$, che è effettivamente trascurabile rispetto a δy_j nell'esperimento qui considerato.

[5] Osservate che il best-fit compiuto è, di fatto, del tipo ai minimi quadrati. Infatti l'incertezza sulla "variabile indipendente" è trascurabile e quella sulla "variabile dipendente" è costante per tutti i dati.

[6] L'eventuale discrepanza di a dal valore "massimo" 1023 potrebbe essere dovuto alla non completa sincronia tra partenza delle fasi di carica o scarica e partenza della misura dei tempi.

[7] L'incertezza sui parametri di best-fit è minore per i fit a due parametri. Questo è atteso considerando la presenza di covarianze negative.

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

[9] Il tempo caratteristico τ_S risulta maggiore per l'analisi dei dati smoothed: questa è una ovvia conseguenza del processo di integrazione temporale introdotto dall'operazione di smoothing.