Walter Ruggeri (wruggeri)

UN AUSILIO INFORMATICO PER L'ELETTRONICO CHE HA POCO TEMPO

25 February 2017

INTRODUZIONE

La situazione la conosciamo tutti: hai davanti un circuito da risolvere, ti armi di buona volontà, scrivi le equazioni ai nodi... e ti rendi conto di avere davanti un sistema di ben 20 equazioni! Altro che buona volontà, qui si parla di voto al martirio! È in casi come questi (ed anche in altri molto meno banali ma altrettanto fastidiosi) che il progettista deve farsi furbo: davanti a lui, vicino a lui... insomma, da qualche parte entro il suo raggio visivo ci sarà sicuramente un computer. E cosa è progettato per fare un computer, se non i calcoli (nei quali, tra l'altro, dicono sia piuttosto bravo)? Ovviamente, però, occorre che il progettista sappia parlarci, con il suo computer: sappiamo tutti che dire al computer "senti, coso, ora mi calcoli 'sto sistema" tendenzialmente non funziona... come fare? Una soluzione molto semplice è parlargli in MATLAB: lui di solito quel linguaggio lo capisce (o se non altro sa tradurlo in qualcosa che capisce)! E proprio di questo parleremo oggi: come dire al computer di risolvere alcuni calcoli per noi. Prenderemo in esame alcune situazioni a basso livello di complessità, e vedremo come ottenere il nostro risultato in maniera rapida ed efficace.

IL LINGUAGGIO MATLAB

***DISCLAIMER**: quanto segue non vuole essere una guida esaustiva sulla programmazione in MATLAB, nè intende proporre uno stile di programmazione professionale o in alcun modo raffinato: unico obiettivo della trattazione è fornire al "progettista al lavoro" alcune idee per scrivere velocemente script che possano aiutarlo nello svolgimento delle sue mansioni.*

Prima di cominciare, una breve spiegazione: MATLAB è un ambiente di calcolo supportato da un suo proprio ed omonimo linguaggio, il quale permette la scrittura di comandi che svolgano opportune operazioni matematiche (generazione di matrici, risoluzione di equazioni differenziali, fattorizzazione di polinomi, trasformate...). Precisamente, MATLAB è orientato al calcolo numerico (per il quale prevede un'estesa libreria di funzioni), ma permette il calcolo simbolico ed incorpora moltissime funzioni di vario genere (di solito racchiuse in *toolboxes*). La suite MATLAB è a pagamento, per cui è possibile che non tutti siano in condizioni di usarla; in tal caso, si può provvedere dotandosi di alternative gratuite come GNU Octave, che supportano gran parte delle funzioni di MATLAB. Si noti, comunque, che quello che analizzeremo in questo articolo è tutto codice scritto e testato su MATLAB R2016a, e non si garantisce la compatibilità totale con

altri software. Detto questo, procediamo con una veloce panoramica sulla sintassi di MATLAB; informazioni più approfondite in merito si possono trovare nel primo capitolo di [1] e soprattutto in [3].

DATI IN MATLAB

Diciamo subito che il tipo di dato elementare è la matrice: MATLAB ragiona ed opera su matrici (sapete da cosa deriva, a proposito, il nome MATLAB?), ovvero considera anche i dati scalari come matrici (di dimensione 1x1, ovviamente). La dichiarazione di un dato è effettuata nella forma

```
A = [a11  a12 ... a1m; ... ; an1 an2 ... anm]
```

Che permette di ottenere la matrice $A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$

Ovviamente, per dati scalari si può anche usare la definizione classica $x = n$. Gli elementi di una matrice sono raggiungibili con la scrittura

```
matrice(i, j, k, ...)
```

Dove gli indici partono da 1.
In MATLAB, è possibile definire delle espressioni simboliche nella forma $f = f(x)$, avendo però cura di dichiarare prima la variabile indipendente con l'apposito comando

```
syms var;
```

Quindi, per definire ad esempio la funzione di Runge $f(x) = \frac{1}{1 + 25x^2}$ possiamo usare la scrittura:

```
syms x;
f = 1/(1 + 25*x^2);
```

FUNZIONI IN MATLAB

In MATLAB, ogni funzione è definita nella forma [...] = *func*(...), con tra parentesi quadre i nomi dei dati che ritorna e tra parentesi tonde i parametri che riceve. Di solito, la scrittura di funzioni proprie avviene su file con lo stesso nome della funzione. Ciascuna funzione ha la struttura

```
[a, b, c, ...] = func (...)
     ...
     return
end
```

Alcune funzioni notevoli di interesse generale sono:

FUNZIONE	SPIEGAZIONE
abs(x)	Modulo
rem(x, y)	Resto di x/y
conj(x)	Complesso coniugato
cos(x), acos(x), ...	Funzioni goniometriche
det(x)	Determinante
size(a)	Dimensioni della matrice
inv(a)	Matrice inversa

Ed è possibile avere informazioni su ciascuna funzione (che di solito esiste in più forme: MATLAB sfrutta pesantemente l'*overloading*) utilizzando il comando

```
help funzione
```

CONDIZIONI E CONTROLLO IN MATLAB

MATLAB supporta i principali costrutti:

```
if condizione
     ...
else if condizione
     ...
else
     ...
end


while condizione
     ...
end


for var = valori
     ...
end
```

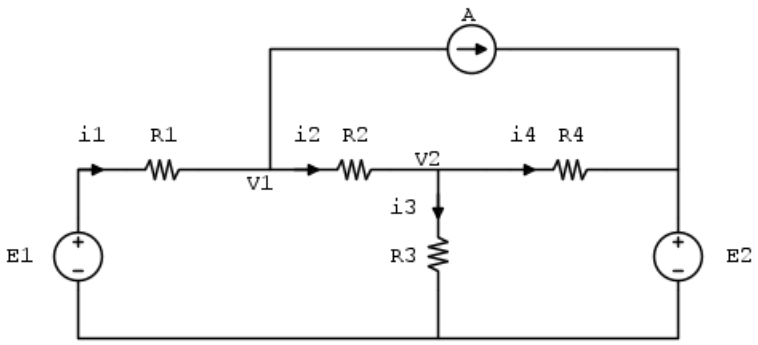
Per la definizione delle condizioni, sono disponibili i classici operatori

OPERATORE	SIGNIFICATO
==	Uguaglianza
~=	Disuguaglianza
>, <, >=, <=	Relazioni d'ordine
&, and(a, b)	AND logico
or(a, b)	OR logico
~, not(a)	NOT logico

ESEMPI DI UTILIZZO

PRIMO ESEMPIO: RISOLUZIONE DI UN CIRCUITO

Bene, cominciamo con il primo esempio: dobbiamo risolvere il semplice circuito rappresentato, tratto da un esercizio di [2]:



Dove

$R_1 = 3\Omega$

$R_2 = R_3 = R_4 = 2\Omega$

$E_1 = 30V$

$E_2 = 60V$

$A = 10A$

Come ben sappiamo, possiamo scrivere le equazioni ai nodi così:

$$\begin{cases} \frac{E_1 - V_1}{R_1} = \frac{V_1 - V_2}{R_2} + A \\ \frac{V_1 - V_2}{R_2} = \frac{V_2 - E_2}{R_4} + \frac{V_2}{R_3} \end{cases}$$

Esse, riscritte in forma più consona, permettono di definire la forma matriciale del sistema:

$$\begin{bmatrix} -\frac{1}{R_1} - \frac{1}{R_2} & \frac{1}{R_2} \\ \frac{1}{R_2} & -\frac{1}{R_2} - \frac{1}{R_3} - \frac{1}{R_4} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} A - \frac{E_1}{R_1} \\ -\frac{E_2}{R_4} \end{bmatrix}$$

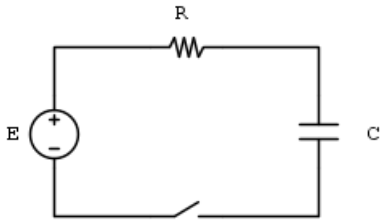
Ora, facciamo intervenire MATLAB, dandogli in pasto il codice:

```
e1 = 30;
e2 = 60;
r1 = 3;
r2 = 2;
r3 = 2;
r4 = 2;
a = 10;

mat = [-1/r1-1/r2 1/r2; 1/r2 -1/r2-1/r3-1/r4];
tn = [a-e1/r1; -e2/r4];

v = inv(mat) * tn
i1 = (e1 - v(1)) / r1
i2 = (v(1) - v(2)) / r2
i3 = v(2) / r3
i4 = (v(2) - e2) / r4
```

Sul quale c'è poco da dire: abbiamo semplicemente utilizzato le scritture viste nella sezione precedente. Da notare solo la risoluzione del sistema nella forma $x = A^{-1}B$, nonchè l'assenza del punto e virgola nelle ultime righe: questa scrittura fa stampare il risultato dei comandi, permettendoci di vedere immediatamente i valori che volevamo.

SECONDO ESEMPIO: VISUALIZZAZIONE DI UN TRANSITORIO

Andiamo al secondo esempio: la visualizzazione di un transitorio. Per questo esempio, dovremo imparare un paio di funzioni, quindi per non complicarci la vita utilizzeremo un circuito molto semplice:



Ora, sappiamo che per un condensatore $i(t) = C\frac{\partial V_c(t)}{\partial t}$, ed inoltre sappiamo che nel nostro circuito $E = Ri(t) + V_c(t)$; questo ci permette di scrivere:

$$\frac{\partial V_c(t)}{\partial t} + \frac{V_c(t)}{RC} = \frac{E}{RC}$$

Per valutare l'andamento di $V_c(t)$, dobbiamo essere in grado di fornire l'equazione differenziale a MATLAB. Come fare?

Intanto, vediamo alcune funzioni utili:

FUNZIONE	DESCRIZIONE
diff(f, x, n)	Derivata n-esima di f rispetto alla variabile x
dsolve(eq, cond, ...)	Risolve l'equazione differenziale eq con le condizioni iniziali indicate
linspace(a, b, n)	Crea un vettore riga con n valori equispaziati in $[a,b]$
plot(vx, vy)	Rappresenta la funzione che passa per i punti indicati
subs(f)	Calcola il valore di una funzione simbolica nel punto indicato dalla sua variabile indipendente

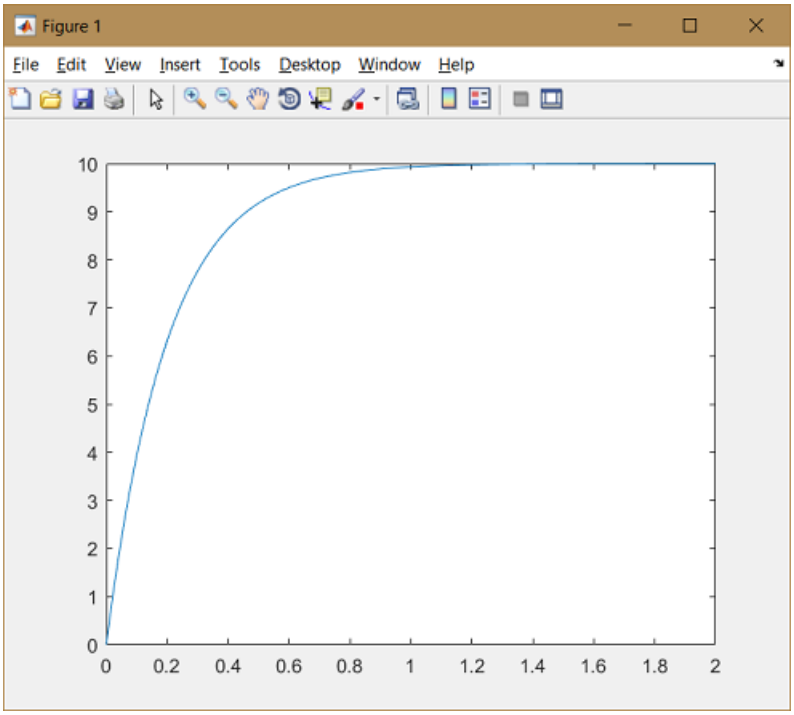
Con queste funzioni (e ponendo $V_c(0) = 0V$, $R = 10\Omega$, $E = 10V$ e $C = 20mF$), possiamo definire la nostra equazione differenziale e rappresentarla con questo codice:

```
syms vc(t);
r = 10;
c = 2e-02;
e = 10;

f = dsolve(diff(vc, t, 1) + vc/(r*c) == e/(r*c), vc(0) == 0);

x = linspace(0, 2, 100);

for i = 1:100
    t = x(i);
    v(i) = subs(f);
```

```
end

plot(x, v);
```

Si notino:

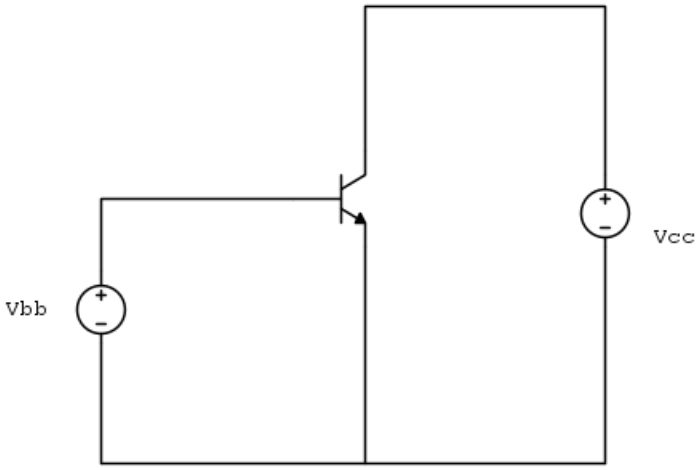
- La forma dell'intervallo nella condizione del ciclo: è una forma di scrittura molto importante in MATLAB, usata in svariate situazioni
- La formulazione delle espressioni simboliche nella funzione di risoluzione
- La definizione della capacità con la notazione esponenziale
- La dichiarazione di una variabile simbolica identificante una funzione, invece che un parametro: la scrittura usata denomina la funzione, ed inoltre ne identifica la variabile



Cosa dobbiamo aspettarci dal nostro script

TERZO ESEMPIO: POLARIZZAZIONE DI UN TRANSISTOR

Questo terzo esempio ci porterà all'analisi dell'ipotetico comportamento di un transistor bipolare NPN polarizzato dal circuito (tratto da un esempio proposto in [4]):



Per ottenere ciò che vogliamo, daremo a MATLAB le equazioni di Ebers-Moll come riportate sempre in [4]:

$$I_c = I_s\left[e^{\frac{V_{be}}{V_T}} - e^{\frac{V_{bc}}{V_T}}\right] - \frac{I_s}{\beta_r}\left[e^{\frac{V_{bc}}{V_T}} - 1\right]$$
$$I_e = I_s\left[e^{\frac{V_{be}}{V_T}} - e^{\frac{V_{bc}}{V_T}}\right] + \frac{I_s}{\beta_f}\left[e^{\frac{V_{be}}{V_T}} - 1\right]$$
$$I_b = \frac{I_s}{\beta_r}\left[e^{\frac{V_{bc}}{V_T}} - 1\right] + \frac{I_s}{\beta_f}\left[e^{\frac{V_{be}}{V_T}} - 1\right]$$

Con queste equazioni, analizzeremo l'evoluzione delle tre correnti al variare di $V_{bb} = V_{be}$, ponendo
$V_{cc} = V_{ce} = 5V$
$I_s = 10^{-16}A$
$\beta_f = 50$
$\beta_r = 1$

Il codice che utilizzeremo non presenta alcunchè che non abbiamo già visto (a parte una funzione per il disegno di più grafici in una sola finestra finestra ed una per la creazione della finestra stessa, delle quali facciamo un uso talmente intuitivo da rendere ogni approfondimento pura pedanteria), ed è precisamente il seguente:

```
syms vbe;
vcc = 5;
is = 1e-16;
bf = 50;
br = 1;
vt = 25e-03;
vce = vcc;
```

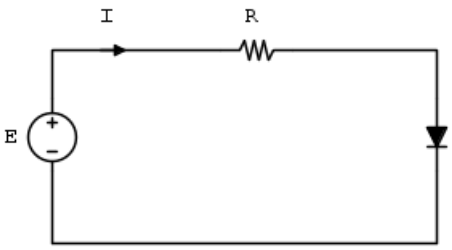
```
vbc = vbe - vce;

ic = is * (exp(vbe/vt) - exp(vbc/vt)) - is/br * (exp(vbc/vt) - 1);
ie = is * (exp(vbe/vt) - exp(vbc/vt)) + is/bf * (exp(vbe/vt) - 1);
ib = is/bf * (exp(vbe/vt) - 1) + is/br * (exp(vbc/vt) - 1);

vbb = linspace(0, 1, 100);

for i = 1:100
    vbe = vbb(i);
    vic(i) = subs(ic);
    vie(i) = subs(ie);
    vib(i) = subs(ib);
end

figure();
subplot(3, 1, 1);
plot(vbb, vic);
subplot(3, 1, 2);
plot(vbb, vie);
subplot(3, 1, 3);
plot(vbb, vib);
```

QUARTO ESEMPIO: PUNTO DI LAVORO DI UN DIODO

Poteva mancare un esempio di calcolo numerico? Ovviamente no! Quindi, andiamo subito al punto: abbiamo un diodo, e ci interessa determinare il suo punto di lavoro in continua... per farlo con carta e penna, dovremmo accettare l'inevitabile somma di incertezze non di rado importanti:

- Non siamo in grado di disegnare con precisione la caratteristica del diodo
- Non siamo in grado di calcolare il punto di intersezione della caratteristica con la retta di carico
- Non siamo in grado di determinare visivamente un'approssimazione molto precisa di tale punto

In casi del genere, ci possono venire in aiuto gli algoritmi di risoluzione approssimata delle equazioni. Per una trattazione opportuna di tali metodi, si rimanda a [5] (che include anche interessanti implementazioni dei relativi algoritmi in MATLAB); noi useremo, per semplicità, il metodo di bisezione. Dato il circuito



Possiamo scrivere l'equazione:

$$I_s(e^{\frac{V_D}{V_T}} - 1) - \frac{E - V_D}{R} = 0$$

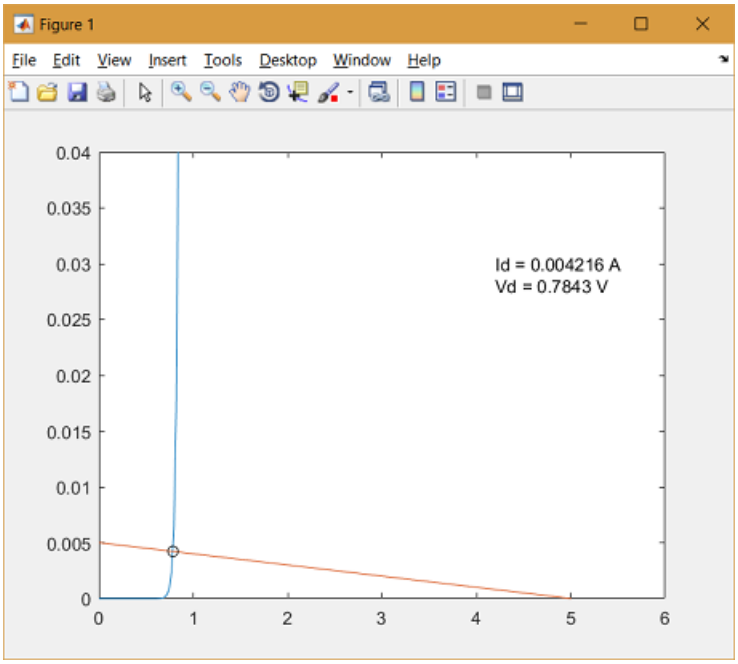
E assegnando i valori (che prendiamo da un esempio proposto in [6]) alle nostre grandezze:

$E = 5V$
$R = 1k\Omega$
$I_s = 10^{-16}A$

Possiamo ottenere la soluzione che cerchiamo:

```
syms vd;
r = 1e3;
e = 5;
is = 1e-16;
vt = 25e-03;
n_passi = 100;

id = is * (exp(vd/vt) - 1);
ir = (e - vd)/r;
f = id - ir;

vdgraph = linspace(0, 6, 250);

vdlow = 0;
vdmid = e/2;
vdhigh = e;
for i = 1:n_passi
    vd = vdlow;
    flow = subs(f);
    vd = vdmid;
    fmid = subs(f);

    if((flow * fmid) < 0)
```

```
        vdhigh = vdmid;
    else
        vdlow = vdmid;
    end

     vdmid = (vdhigh + vdlow) / 2;
end

for i = 1:250
    vd = vdgraph(i);
    idgraph(i) = subs(id);
    irgraph(i) = subs(ir);
end

vd = vdmid;
vdfinal = vpa(vd, 4);
idfinal = vpa(subs(id), 4);

plot(vdgraph, idgraph, vdgraph, irgraph, vdfinal, idfinal, '-ok');
axis([0 6 0 0.04]);
txt = ['Id = ', char(idfinal), ' A'];
txt2 = ['Vd = ', char(vdfinal), ' V'];
text(4.2, 0.03, txt);
text(4.2, 0.028, txt2);
```



Output dello script

Nel codice, abbiamo usato alcune funzioni "nuove", che vediamo qui elencate e spiegate:

FUNZIONE	DESCRIZIONE
vpa(sym, d)	Converte un valore simbolico in un floating point con d cifre significative
axis([xmin xmax ymin ymax])	Imposta gli estremi degli assi del grafico
text(x, y, txt)	Scrive il testo alle coordinate indicate
char(val)	Converte un valore numerico in una stringa

Si notino inoltre la creazione delle stringhe come veri e propri vettori di caratteri e l'utilizzo, per disegnare il grafico, di appositi caratteri (definiti *LineSpecs*) per specificare lo stile di rappresentazione del punto di lavoro.

A margine, una piccola nota per i più curiosi: ovviamente, nella sua estesa libreria MATLAB prevede una funzione in grado di risolvere da sè l'equazione, ed è la funzione

```
solve(eqn, var)
```

Tuttavia, si è scelto di procedere comunque alla scrittura dell'algoritmo, dato che presentare il banale utilizzo dell'apposita funzione non avrebbe avuto alcun valore didattico.

CONCLUSIONI

Abbiamo visto alcuni piccoli esempi di ciò che si può fare con MATLAB; ovviamente, avremmo potuto vedere tanto altro: analisi in frequenza, progetto di filtri e controllori, analisi da schema a blocchi... davvero, MATLAB può svolgere tantissimi compiti utili. Per vedere tutto questo, però, avremmo dovuto appesantire la trattazione con opportuni rimandi teorici ed un'opportuna analisi del linguaggio di programmazione (qui ridotta davvero al minimo necessario)... di fatto, l'articolo avrebbe dovuto assumere un tono molto meno leggero del voluto.
Naturalmente, gli esempi proposti sono tutti banali e facilmente risolvibili anche a mano... ma da un articolo introduttivo e propositivo, sarebbe stupido aspettarsi altrimenti: di solito, chi si avvicina per la prima volta ad un linguaggio di programmazione trova conforto nella possibilità di poter verificare con carta e penna la corrispondenza tra il codice scritto e gli algoritmi che voleva implementare.

BIBLIOGRAFIA

[1] R.V. Dukkipati: "MATLAB: an introduction with applications", New Age, 2010
[2] R. Perfetti: "Circuiti elettrici", Zanichelli, 2013
[3] Documentazione ufficiale di MATLAB
[4] R.C. Jaeger, T.N. Blalock: "Microelectronic circuit design", McGraw-Hill, 2016
[5] G. Monegato: "Metodi e algoritmi per il calcolo numerico", CLUT, 2008
[6] N. Storey: "Electronics: a systems approach", Pearson, 2013