"Quaderni di Ricerca in Didattica (Mathematics)", n. 27, 2017

G.R.I.M. (Department of Mathematics, University of Palermo, Italy)

L'algoritmo di Euclide generalizzato e la soluzione di equazioni diofantee lineari

Massimo Salvi

massimo.salvi1@istruzione.it

Abstract: *In this paper we describe an algorithm to solve the linear diophantine equation* $a_1x_1 + a_2x_2 + ... + a_nx_n = n$ *which generalizes the Bachet-Bézout identity to more than two unknowns. The proposed method is possibly useful for teaching in secondary schools with the help of computer programming. The method is based on the classical euclidean division algorithm that we generalize to more than two integers by using matrices. The algorithms that you can find in literature for solving this problem, are usually based on the iteration of euclidean division algorithm applied on two integers by the so called Eulero's method [1]. The algorithm that we propose uses matrix operations that are easy to be implemented using a programming language as we propose in appendix. We propose such an algorithm in a form different from the ones proposed by other authors [2].*

Riassunto: *In questo articolo viene descritto un algoritmo per la risoluzione dell'equazione diofantea lineare* $a_1x_1 + a_2x_2 + ... + a_nx_n = n$ *che generalizza la cosiddetta identità di Bachet-Bézout a più variabili. Il metodo proposto si presta ad un percorso da proporre alle scuole superiori con l'ausilio della programmazione. Il procedimento è basato sull'algoritmo di divisione euclidea generalizzato che viene considerato anche in forma matriciale. In genere gli algoritmi proposti per la risoluzione di queste equazioni si basano su un uso iterato della divisione euclidea applicata a due coefficienti con il metodo di Eulero [1]. L'algoritmo che proponiamo utilizza operazioni su matrici e ben si presta ad una implementazione in un linguaggio di programmazione che proponiamo in appendice. Tale algoritmo viene proposto in una forma diversa da altri algoritmi precedentemente proposti da altri autori e basati su un'idea simile [2].*

Subject Classification: 97Fxx, 68Wxx, 11Axx, 11Dxx

1. L'algoritmo euclideo generalizzato

L'idea che sta alla base dell'algoritmo euclideo della divisione ([3], p. 23), che permette di determinare il massimo comune divisore fra due interi, può essere generalizzata a insiemi di più di due interi (vedere es. [4], [5]).

Noi proponiamo il seguente algoritmo; dato un insieme di interi, ipotizziamoli non nulli, che possiamo pensare come costituenti un vettore riga $C = (a_1 \quad a_2 \quad ... \quad ... \quad a_n)$, consideriamo la seguente successione di operazioni :

1. si cambia il segno di tutti i coefficienti negativi rendendoli positivi (questo non cambia il valore del massimo comune divisore);
2. si trova il minimo fra i coefficienti;
3. iniziando dal primo coefficiente diverso da zero e diverso dal coefficiente che occupa la posizione del minimo, si calcola il resto della divisione del coefficiente stesso per il minimo;
4. se il resto è uguale a 0 si sostituisce tale valore al coefficiente e si procede con un il coefficiente successivo;
5. se il resto è diverso da 0, si sostituisce al coefficiente tale valore, si considera tale valore come il nuovo valore di minimo e si riparte dal punto 3;
6. si ripetono le operazioni dal punto 3 fino a quando resta un solo coefficiente diverso da 0.

Teorema 1. Al termine del precedente algoritmo, il coefficiente diverso da 0 che rimane nel vettore è il massimo comune divisore fra i coefficienti iniziali del vettore C.

Dimostrazione.

Se indichiamo con m il minimo valore dei coefficienti e posto, per un determinato a_i diverso dal coefficiente che è il minimo, $a_i = qm + r$, è' sufficiente dimostrare che $mcd(a_1 \quad .. \quad a_i \quad a_n) = mcd(a_1 \quad .. \quad r \quad a_n)$. Poniamo $g = mcd(a_1 \quad .. \quad a_i \quad a_n)$, tale numero sarà un divisore anche dell'insieme dei nuovi coefficienti $(a_1 \quad .. \quad r \quad a_n)$, infatti tutti i coefficienti tranne a_i sono invariati e il nuovo coefficiente (il resto della divisione) r è divisibile per g, dato che vale $r = a_i - qm$ e sia a_i che m (che è uno dei coefficienti) sono, per ipotesi, divisibili per g.

Se ipotizziamo, per assurdo, che esista un numero g' divisore dei nuovi coefficienti tale che g'>g , allora si avrebbe che tale numero sarebbe un divisore anche di a_i, infatti $a_i = qm + r$, ma sia m che r sono divisibili per g' pertanto lo sarà anche a_i. Da questo segue che g non può essere il massimo comune divisore dei coefficienti iniziali contraddicendo l'ipotesi iniziale che lo fosse, pertanto resta dimostrato che g sarà il massimo comune divisore anche dei nuovi coefficienti.

Ad ogni passaggio il valore dei coefficienti diminuisce e l'algoritmo si arresta quando resta un solo valore diverso da 0 che sarà il massimo comune divisore dei coefficienti iniziali.
Osserviamo che al punto 3 se ci sono altri valori uguali al minimo individuato, verranno sostituiti con uno 0.
Come si vede, l'algoritmo applicato ad un vettore con soli due coefficienti coincide con il noto algoritmo euclideo.

2. Forma matriciale dell'algoritmo euclideo generalizzato

Mostriamo che l'algoritmo descritto nel paragrafo precedente è equivalente alla moltiplicazione a sinistra del vettore colonna C^T per una certa matrice quadrata a coefficienti interi M di dimensione $n \times n$. Tale matrice si può costruire considerando che ogni operazione sui coefficienti (vettore C) effettuata nell'algoritmo è equivalente alla moltiplicazione a sinistra per una matrice quadrata a coefficienti interi di dimensione $n \times n$, infatti:

a. cambiare il segno di uno dei coefficienti è equivalente a moltiplicare a sinistra il vettore colonna C^T per una matrice del tipo : $\begin{bmatrix} 1 & .. & .. & 0 \\ .. & .. & & 0 \\ & & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ ovvero una matrice sulla cui diagonale ci sono tutti 1 tranne un -1 (relativo alla posizione del coefficiente il cui segno va cambiato) e tutti gli altri valori sono 0. Il determinante di tale tipo matrice vale -1.

b. sostituire un coefficiente a_i del vettore con il resto della divisione del coefficiente per m (che è uno dei coefficienti di C coincidente con minimo dei coefficienti) è equivalente a moltiplicare a sinistra il vettore colonna C^T per una matrice del tipo: $\begin{bmatrix} 1 & .. & .. & 0 \\ .. & .. & & 0 \\ & -q & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ ovvero una matrice che corrisponde alla trasformazione che lascia invariati tutti i coefficienti tranne a_i che viene sostituito con $r = a_i - qm$, in cui r e q sono rispettivamente il resto e il quoto della divisione fra a_i e m. Il determinante di tale tipo di matrice vale 1.

Dalle precedenti considerazioni segue che l'algoritmo, applicando in sequenza le trasformazioni del tipo suddetto, è equivalente alla moltiplicazione a sinistra del vettore C^T di una unica matrice M data dal prodotto

delle diverse matrici relative alle singole trasformazioni. Per comodità al termine del precedente algoritmo possiamo scambiare la posizione dei coefficienti finali del vettore C in modo che il massimo comune divisore occupi la prima posizione del vettore e gli altri siano tutti nulli. Questa operazione equivale a moltiplicare a sinistra il vettore C^T per una matrice quadrata, anch'essa di dimensione $n \times n$, che esegue lo scambio fra due elementi del vettore C^T. Tale matrice è del tipo $\begin{bmatrix} 0 & 1 & .. & 0 \\ 1 & 0 & & 0 \\ & & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. Il determinante di tale tipo di matrice vale -1.

Osserviamo che la matrice M, sfruttando il teorema di Binet, ha determinante che vale 1 o -1, dato che è ottenuta dal prodotto di matrici che hanno determinante 1 oppure -1 (si tratta di trasformazioni unimodulari).

3. Risoluzione dell'equazione: $a_1x_1 + a_2x_2 + ... + a_nx_n = d$

Da quanto discusso fino a questo punto, dato il vettore C dei coefficienti dell'equazione

(*1) $a_1x_1 + a_2x_2 + ... + a_nx_n = d$, possiamo costruire , sfruttando l'algoritmo euclideo generalizzato, una matrice M tale che : $M \begin{bmatrix} a_1 \\ a_2 \\ .. \\ a_n \end{bmatrix} = \begin{bmatrix} mcd(a_1,..,a_n) \\ 0 \\ .. \\ 0 \end{bmatrix}$ (*2)

Questa equazione matriciale corrisponde a un sistema di n equazioni la prima delle quali ci fornisce una soluzione particolare dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$, mentre le rimanenti sono soluzioni particolari dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = 0$. A questo punto siamo in grado di enunciare il seguente e noto [6] :

Teorema 2. L'equazione diofantea $a_1x_1 + a_2x_2 + ... + a_nx_n = d$ è risolvibile se e solo se *d* è divisibile per $mcd(a_1,..a_n)$.

Dimostrazione.

La condizione è necessaria, infatti, ammesso che l'equazione abbia una soluzione $a_1x'_1 + a_2x'_2 + ... + a_nx'_n = d$, indicando con g il massimo comune divisore dei coefficienti, si può scrivere

$g(a'_1 x'_1 + a'_2 x'_2 + ... + a'_n x'_n) = d$, in cui i nuovi coefficienti sono gli interi $a_i' = \frac{a_i}{g}$, quindi d deve essere divisibile per g.

La sufficienza della condizione segue dall'equazione (*2), che fornisce una soluzione particolare nel caso che d sia uguale a $mcd(a_1,..a_n)$. Se d è un multiplo di $mcd(a_1,..a_n)$, ponendo $d = k \cdot mcd(a_1,..a_n)$ si ottiene una soluzione moltiplicando per k ciascuno degli x_i che risolvono l'equazione nel caso $d = mcd(a_1,..a_n)$.

Osserviamo che il teorema vale anche nel caso d=0. Infatti in tal caso 0 è sempre divisibile per $mcd(a_1,..a_n)$ e una soluzione esiste sempre ed è data da $x_i = 0 \ \forall i$.

Se consideriamo l'identità di Bachet-Bézout $a_1x_1 + a_2x_2 + ... + a_nx_n = 1$, il precedente teorema permette di concludere che tale equazione è risolvibile se e solo se i coefficienti sono primi fra loro.

Partendo dall'equazione (*2) siamo in grado di trovare tutte le soluzioni dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = d$ quando $d = mcd(a_1,..a_n)$.

Se definiamo il vettore riga $X = \begin{pmatrix} x_1 & x_2 & ... & ... & x_n \end{pmatrix}$ e indichiamo con R_i la i-esima riga della matrice M, vale il seguente:

Teorema 3. Tutte e sole le soluzioni dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$ sono date da $X = R_1 + \lambda_2 R_2 + .. + \lambda_n R_n$, (*3) dove i parametri λ_i possono assumere tutti i possibili valori interi.

Dimostrazione.

Dalla linearità dell'equazione segue facilmente che per ogni insieme di valori dei parametri si ottiene una soluzione di $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$, infatti se esplicitiamo $M = \begin{bmatrix} m_{11} & m_{12} & .. & m_{1n} \\ m_{21} & m_{22} & & \\ .. & & .. & \\ m_{n1} & & & m_{nn} \end{bmatrix}$, possiamo scrivere :

$$X = R_1 + \lambda_2 R_2 + .. + \lambda_n R_n = \begin{pmatrix} m_{11} + \lambda_2 m_{21} + .. + \lambda_n m_{n1} & m_{12} + \lambda_2 m_{22} + .. + \lambda_n m_{n2} & ... & m_{1n} + \lambda_2 m_{2n} + .. + \lambda_n m_{nn} \end{pmatrix}$$

e sostituendo i valori trovati nell'equazione (*1) e considerando le equazioni (*2) si ottiene:

$$a_1\left(m_{11}+\lambda_2 m_{21}+..+\lambda_n m_{n1}\right)+a_2\left(m_{12}+\lambda_2 m_{22}+..+\lambda_n m_{n2}\right)+..+a_n\left(m_{1n}+\lambda_2 m_{2n}+..+\lambda_n m_{nn}\right)=$$
$$=a_1 m_{11}+a_2 m_{12}+..+a_n m_{1n}+...\lambda_2\left(a_1 m_{21}+a_2 m_{22}+..+a_n m_{2n}\right)+...+\lambda_n\left(a_1 m_{n1}+a_2 m_{n2}+..+a_n m_{nn}\right)=$$

$$=d+...\lambda_2\cdot 0+...+\lambda_n\cdot 0=d$$

Resta da dimostrare che, data una soluzione qualunque di $a_1x_1+a_2x_2+...+a_nx_n=mcd(a_1,..a_n)$, tale soluzione è ottenibile dall'espressione (*3) con opportuni valori interi dei parametri λ_i. Indichiamo con il vettore $X'=\left(x'_1 \quad x'_2 \quad ... \quad ... \quad x'_n\right)$ l'insieme dei valori di una soluzione particolare, vogliamo dimostrare che tale vettore è esprimibile nella forma $X'=R_1+\lambda_2R_2+..+\lambda_nR_n$ con particolari valori interi dei parametri λ_i. A questo scopo è conveniente, dopo avere introdotto un parametro aggiuntivo λ_1, scrivere la precedente relazione sotto forma di sistema (usando la matrice trasposta di M):

$$\begin{bmatrix} m_{11} & m_{21} & .. & m_{n1} \\ m_{12} & m_{22} & & \\ .. & & .. & \\ m_{1n} & & & m_{nn} \end{bmatrix}\begin{bmatrix}\lambda_1\\ \lambda_2\\ ..\\ \lambda_n\end{bmatrix}=\begin{bmatrix}x_1'\\ x_2'\\ ..\\ x_n'\end{bmatrix}$$

In tale sistema i parametri λ_i sono le incognite da determinare e i valori x_i' sono i termini noti di ciascuna equazione. Dobbiamo mostrare che i parametri λ_i esistono e sono interi e che $\lambda_1=1$. Usando il metodo di Cramer [7] possiamo calcolare il generico λ_i:

$$\lambda_i=\frac{\det\begin{bmatrix} m_{11} & m_{21} & x_1' & m_{n1} \\ m_{12} & m_{22} & x_2' & \\ .. & & .. & \\ m_{1n} & & x_n' & m_{nn} \end{bmatrix}}{\det\begin{bmatrix} m_{11} & m_{21} & .. & m_{n1} \\ m_{12} & m_{22} & & \\ .. & & .. & \\ m_{1n} & & & m_{nn} \end{bmatrix}}$$

al denominatore il determinante di M^T è uguale al determinante di M [7] che, come visto in §1 vale 1 o -1, pertanto i valori λ_i esistono e sono interi. Per concludere la dimostrazione è sufficiente sostituire le espressioni che forniscono le soluzioni nell'equazione (*1):

$$a_1\left(m_{11}+\lambda_2 m_{21}+..+\lambda_n m_{n1}\right)+a_2\left(m_{12}+\lambda_2 m_{22}+..+\lambda_n m_{n2}\right)+..+a_n\left(m_{1n}+\lambda_2 m_{2n}+..+\lambda_n m_{nn}\right)=$$

$$= \lambda_1(a_1m_{11} + a_2m_{12} + .. + a_nm_{1n}) + ...\lambda_2(a_1m_{21} + a_2m_{22} + .. + a_nm_{2n}) + ... + \lambda_n(a_1m_{n1} + a_2m_{n2} + .. + a_nm_{nn}) =$$

$$= \lambda_1 d + ...\lambda_2 \cdot 0 + ... + \lambda_n \cdot 0 = d$$

quindi si ottiene $\lambda_1 d = d$ e, dato che supponiamo $d \neq 0$, deve essere $\lambda_1 = 1$.

Naturalmente l'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$ può essere ricondotta all'equazione equivalente $a_1'x_1 + a_2'x_2 + ... + a_n'x_n = 1$ (identità di Bachet-Bézout) con i coefficienti primi fra loro, dividendo i due membri per $mcd(a_1,..a_n)$.

4. Un esempio

Applichiamo il metodo descritto per risolvere l'equazione $2x_1 - 7x_2 + 4x_3 = 1$

Nella seguente tabella evidenziamo le trasformazioni operate sul vettore dei coefficienti utilizzando l'algoritmo, le relative matrici e la matrice finale M ottenuta moltiplicando tutte le singole matrici.

cambio del segno	$c_2 \leftarrow c_2 - 3c_1$	$c_1 \leftarrow c_1 - 2c_2$	$c_3 \leftarrow c_3 - 4c_2$
$\begin{bmatrix}1&0&0\\0&-1&0\\0&0&1\end{bmatrix}\begin{bmatrix}2\\-7\\4\end{bmatrix}=\begin{bmatrix}2\\7\\4\end{bmatrix}$	$\begin{bmatrix}1&0&0\\-3&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}2\\7\\4\end{bmatrix}=\begin{bmatrix}2\\1\\4\end{bmatrix}$	$\begin{bmatrix}1&-2&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}2\\1\\4\end{bmatrix}=\begin{bmatrix}0\\1\\4\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&1&0\\0&-4&1\end{bmatrix}\begin{bmatrix}0\\1\\4\end{bmatrix}=\begin{bmatrix}0\\1\\0\end{bmatrix}$
inversione delle righe	matrice M		
$\begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}=\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}-3&-1&0\\7&2&0\\12&4&1\end{bmatrix}=\begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&-4&1\end{bmatrix}\begin{bmatrix}1&-2&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\-3&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\0&-1&0\\0&0&1\end{bmatrix}$		

Abbiamo che mcd(2,-7,4)=1, quindi l'equazione è risolvibile. Utilizzando il Teorema 3 possiamo ricavare tutte le soluzioni:

$$(x_1 \quad x_2 \quad x_3) = (-3 \quad -1 \quad 0) + \lambda_2 \cdot (7 \quad 2 \quad 0) + \lambda_3 \cdot (12 \quad 4 \quad 1)$$

oppure $\begin{cases} x_1 = -3 + 7\lambda_2 + 12\lambda_3 \\ x_2 = -1 + 2\lambda_2 + 4\lambda_3 \\ x_3 = \lambda_3 \end{cases}$ in cui i parametri λ_2 e λ_3 possono assumere tutti i valori interi

5. Il caso generale

Il Teorema 3 è il punto di partenza per trovare tutte le soluzioni di $a_1x_1 + a_2x_2 + ... + a_nx_n = d$ nei casi in cui *d* è un multiplo di $mcd(a_1 \quad .. \quad a_i \quad a_n)$ (quindi anche zero).

Consideriamo come nel Teorema 3 le soluzioni dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$, che sono date da $X = R_1 + \lambda_2R_2 + .. + \lambda_nR_n$, allora possiamo dimostrare il seguente:

Teorema 4. Tutte e sole le soluzioni dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = 0$ (*4) sono date da $X' = \lambda_2R_2 + .. + \lambda_nR_n$, dove i parametri λ_i possono assumere tutti i possibili valori interi e $R_2,...R_n$ sono definiti come nel precedente Teorema 3.

Dimostrazione.

Cominciamo mostrando che i vettori di interi dati da $X' = \lambda_2R_2 + .. + \lambda_nR_n$ sono soluzioni di $a_1x_1 + a_2x_2 + ... + a_nx_n = 0$. Per quanto dimostrato nel Teorema 3 sappiamo che $X = R_1 + \lambda_2R_2 + .. + \lambda_nR_n$ sono soluzioni di $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$, in particolare R_1 è la soluzione che otteniamo ponendo tutti i $\lambda_i = 0$. Possiamo scrivere $X' = X - R_1$, si vede che X' è una soluzione di *4, infatti se esplicitiamo i due vettori e scriviamo $X = (x_1, x_2,..x_n)$ e $R_1 = (r_1, r_2,..r_n)$, sappiamo che $a_1x_1 + a_2x_2 + ... + a_nx_n = mcd(a_1,..a_n)$ e $a_1r_1 + a_2r_2 + ... + r_nx_n = mcd(a_1,..a_n)$

e quindi, sottraendo membro a membro, possiamo scrivere

$$a_1x_1 + a_2x_2 + ... + a_nx_n - (a_1r_1 + a_2r_2 + ... + a_nr_n) = a_1(x_1 - r_1) + a_2(x_2 - r_2) + .. + a_n(x_n - r_n) = 0$$

che equivale a dire che il vettore $X' = X - R_1$ è una soluzione di $a_1x_1 + a_2x_2 + ... + a_nx_n = 0$.

Ci resta da provare che ogni soluzione di *4 è esprimibile nella forma $X' = \lambda_2R_2 + .. + \lambda_nR_n$ con opportuni valori interi λ_i. A questo proposito consideriamo una soluzione $X' = (x'_1, x'_2,..x'_n)$ di *4, si avrà

$a_1x'_1+a_2x'_2+...+a'_n\,x_n=0$, se definiamo vettore $X=X'+R_1=(x'_1+r_1,x'_2+r_2,..,x'_n+r_n)$, questo sarà una soluzione dell'equazione $a_1x_1+a_2x_2+...+a_nx_n=mcd(a_1,..a_n)$, infatti possiamo sostituire e ottenere

$$a_1(x'_1+r_1)+a_2(x'_2+r_2)+...+a_n(x'_n+r_n)=(a_1x'_1+a_2x'_2+...+a_nx'_n)+(a_1r_1+a_2r_2+...+r_nx_n)=0+mcd(a_1,..a_n)$$

Per il Teorema 3 sappiamo che esistono degli interi λ_i tali che la soluzione trovata si può scrivere $X=R_1+\lambda_2R_2+..+\lambda_nR_n$, a questo punto è sufficiente sostituire in $X=X'+R_1$ per ottenere X' nella forma cercata

$R_1+\lambda_2R_2+..+\lambda_nR_n=X'+R_1$ da cui si ottiene subito $X'=\lambda_2R_2+..+\lambda_nR_n$.

Se consideriamo il caso più generale dell'equazione, ovvero

$a_1x_1+a_2x_2+...+a_nx_n=k\cdot mcd(a_1,..a_n)$ con k intero, dimostriamo il seguente:

Teorema 5. Tutte e sole le soluzioni dell'equazione $a_1x_1+a_2x_2+...+a_nx_n=k\cdot mcd(a_1,..a_n)$ (*5) sono date da $X=kR_1+\lambda_2R_2+..+\lambda_nR_n$, dove i parametri λ_i possono assumere tutti i possibili valori interi e $R_1,..R_n$ sono definiti come nel precedente Teorema 3.

Notiamo che i casi k=1 e k=0 coincidono con i risultati dimostrati rispettivamente nel Teorema 3 e nel Teorema 4.

Dimostrazione

Anche in questo cominciamo mostrando che ogni vettore della forma $X=kR_1+\lambda_2R_2+..+\lambda_nR_n$ fornisce una soluzione dell'equazione *5 . Infatti sappiamo dal Teorema 4 che $X'=\lambda_2R_2+..+\lambda_nR_n$ è una soluzione di $a_1x_1+a_2x_2+...+a_nx_n=0$, quindi quando sostituiamo X nell'equazione *5 tali termini si annullano e l'equazione diventa

$a_1kr_1+a_2kr_2+...+a_nkr_n=k(a_1r_1+a_2r_2+...+a_nr_n)=k\cdot mcd(a_1,..a_n)$ che è verificata dato che il vettore R_1 è una soluzione di $a_1x_1+a_2x_2+...+a_nx_n=mcd(a_1,..a_n)$ e quindi si ha $a_1r_1+a_2r_2+...+a_nr_n=mcd(a_1,..a_n)$.

Ora mostriamo che qualunque soluzione X di *5 è esprimibile nella forma $X = kR_1 + \lambda_2 R_2 + .. + \lambda_n R_n$. Se $X = (x_1, x_2, ..x_n)$ è una soluzione si ha che vale

$a_1 x_1 + a_2 x_2 + ... + a_n x_n = k \cdot mcd(a_1, ..a_n)$, ma sappiamo anche che vale

$a_1 r_1 + a_2 r_2 + ... + a_n r_n = mcd(a_1, ..a_n)$ e quindi $a_1 kr_1 + a_2 kr_2 + ... + a_n kr_n = k \cdot mcd(a_1, ..a_n)$

se sottraiamo membro a membro si ottiene

$$a_1 x_1 + a_2 x_2 + ... + a_n x_n - (a_1 kr_1 + a_2 kr_2 + ... + a_n kr_n) = a_1 (x_1 - kr_1) + a_2 (x_2 - kr_2) + .. + a_n (x_n - kr_n) = 0$$

che equivale a dire che il vettore $X - kR_1$ è una soluzione di $a_1 x_1 + a_2 x_2 + ... + a_n x_n = 0$.

Per il Teorema 4 possiamo dire che esistono degli interi λ_i tali che si può scrivere $X - kR_1 = \lambda_2 R_2 + .. + \lambda_n R_n$ da cui si ottiene infine $X = kR_1 + \lambda_2 R_2 + .. + \lambda_n R_n$.

6. Un algoritmo di risoluzione per l'equazione $a_1 x_1 + a_2 x_2 + ... + a_n x_n = n$

Sfruttando i teoremi precedenti siamo in grado di proporre un algoritmo per l'equazione generale $a_1 x_1 + a_2 x_2 + ... + a_n x_n = n$ di cui sintetizziamo i passi e proponiamo in Appendice una possibile implementazione in linguaggio di programmazione C++.

1. Dati i coefficienti dell'equazione $a_1, ..., a_n$ utilizziamo l'algoritmo descritto in §2 per ottenere sia il massimo comune divisore di tali coefficienti che i vettori riga $R_1, ..., R_n$
2. Se il termine noto n dell'equazione non è divisibile per il m.c.d. l'equazione non è risolvibile, al contrario è risolvibile e quindi procediamo a calcolare il valore $k = \dfrac{n}{mcd(a_1, ..., a_n)}$
3. Tutte le soluzioni dell'equazione sono date dal vettore $X = kR_1 + \lambda_2 R_2 + .. + \lambda_n R_n$ in cui compare il *k* calcolato al punto 2, i vettori $R_1, ..., R_n$ trovati al punto 1 e i parametri interi λ_i.

7. Note conclusive

In due precedenti articoli abbiamo già proposto un modo geometrico per dimostrare l'identità di Bachet-Bézout in due incognite, evidenziandone le potenzialità didattiche ([8], [9]). Abbiamo trovato che per passare a più incognite risulta conveniente un metodo algebrico basato sulle matrici. Tale metodo ha,

secondo noi, anche una valenza didattica utilizzabile negli ultimi anni della scuola superiore dove sono previsti lo studio delle matrici e della programmazione (coding). Inoltre ci sembra interessante mostrare come l'algebra delle matrici si possa sfruttare utilmente anche nell'ambito dell'aritmetica superiore.

Riferimenti bibliografici

[1] Davenport H. (1994), *Aritmetica superiore:un'introduzione alla teoria dei numeri*, Zanichelli.

[2] Koshy T. (1998). Linear Diophantine equations, linear congruences and matrices, *Math. Gaz.* **82**, 274-277.

[3] Barozzi G.C. (2006), *Aritmetica : un approccio computazionale*, Zanichelli.

[4] Gilbert W.J., Pathria A., Linear Diophantine Equations http://www.math.uwaterloo.ca/~wgilbert/Research/GilbertPathria.pdf .

[5] Rosser B. (1941). A generalization of the Euclidean algorithm to several dimensions, *Duke Mathematical Journal* **27**, 69-74.

[6] Mordell L.J. (1969), *Diophantine Equations*, Academic Press.

[7] Maltsev A.I. (1980), *Fondamenti di algebra lineare*, Editori Riuniti.

[8] Salvi M. (2014). Una dimostrazione geometrica dell'identità di Bachet- Bézout, *Archimede* **1,** 23-29.

[9] M. Salvi and P. Milici (2013). Laboratorio di matematica in classe: due nuove macchine per problemi nel continuo e nel discreto. *Quaderni di Ricerca in Didattica (Mathematics)* **23**, 15–24.

Appendice

Il seguente programma in C++ richiede di introdurre prima il termine noto n dell'equazione $a_1x_1 + a_2x_2 + ... + a_nx_n = n$, successivamente i coefficienti $a_1,...,a_n$, per terminare l'immissione si deve digitare il valore 0. Ad esempio, per risolvere l'equazione $2x_1 - 7x_2 + 4x_3 = 1$ si devono digitare i numeri 1 (invio), 2 (invio), -7 (invio), 4 (invio), 0 (invio).

```
/* PROGRAMMA PER LA RISOLUZIONE DI EQUAZIONI DIOFANTEE LINEARI IN n INCOGNITE  */
/* Autore: Massimo Salvi     massimo.salvi1@istruzione.it */
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <sstream>
using namespace std;
string intToString(int value) {
   stringstream ss;
   ss << value;
   return ss.str();
}
#define DimMax 100 //* dimensione massima del vettore
int Termine;
int Vettore[DimMax];
int NumElementi;
int U [DimMax][DimMax];
string strApp, strInd,  str2;
void Moltiplica ( int T[DimMax][DimMax]);
```

```
int main() {
        int K,i,j, z, quo, intApp,w, IntT;
        int Pos;
        int MaxVet;
        int MinVet;
        int NumMon;
        int Cont, contM;
        int Od [DimMax][DimMax];
        MaxVet=0;
        /*
                                        Ciclo di immissione del termine noto e coefficienti dell'equazione
        */
        //richiede il termine noto n
        K=0;
        cout << "\n Inserire il termine noto dell'equazione'  :";
        cin >> Termine;
        NumMon=1;
        //ciclo di inserimento dei coefficienti
        while (NumMon!=0)
        {
                cout << "\n Inserire il coefficiente "<<K+1<< " dell'equazione '  :";
                cin >> NumMon;
                if (NumMon!=0){
                        Vettore[K]=NumMon;
                }
                K++;
        }
        NumElementi=K-1;
        cout << "\n";
```

```
//* inizializzazione della matrice U
for (i=0;i<NumElementi;i++)     //* inizializzazione della matrice trasformazione come matrice identità
{
        for (j=0;j<NumElementi;j++)
        {
                if (i==j)
                {
                        U [i][j]=1;
                }
                else
                {
                        U [i][j]=0;
                }
        }
}
/*
        Algoritmo Euclideo generalizzato applicato al vettore Vettore contemporanea costruzione della matrice U
*/
//* gli elementi negativi del vettore vengono resi positivi
for (j=0;j<NumElementi;j++)
        {
                if (Vettore[j]<0)
                {
                        U [j][j]=-1;
                        Vettore[j]=-Vettore[j];
                }
        }
MaxVet=0;
for (K=0;K<NumElementi;K++)
```

```
{
        //* determina il numero maggiore dei coefficienti in valore assoluto
        if (Vettore[K]>MaxVet )
                MaxVet=Vettore[K];
}
MinVet=MaxVet;
Pos=0;
//*Cerca il minimo (in valore assoluto) del vettore diverso da zero
for (K=0;K<NumElementi;K++)
{
        if (Vettore[K]<=MinVet & Vettore[K]!=0)
                {
                        MinVet=Vettore[K] ;
                        Pos=K;
                }
}
//* ciclo che termina quando i valori del vettore diversi da zero sono tutti uguali
do
{
        //IntT=0;
        K=0;
        do {
                if (Vettore[K] !=0 && K!=Pos)
                        {
                                intApp=Vettore[K] % MinVet;
                                quo=(Vettore[K]-intApp)/MinVet;
                                Vettore[K]=intApp;
                                //* costruzione della matrice relativa alla trasformazione
```

```
                    for (i=0;i<NumElementi;i++)     //* inizializzazione della matrice trasformazione come
matrice identità
                    {
                        for (j=0;j<NumElementi;j++)
                      {
                            if (i==j)
                                {
                                    Od [i][j]=1;
                                }
                            else
                            {
                                Od [i][j]=0;
                            }
                      }
                    }
                    //* inserimento di un opportuno elemento per ottenere la trasformazione

                    Od[K][Pos]=-quo;
                    Moltiplica(Od);
                    // se il resto della divisione è diverso da zero cambiamo il valore di MinVet e di Pos
                    if (intApp!=0){
                        MinVet=intApp;
                        Pos=K;
                        K=-1;
                    }
                }
            K++;
} while (K!=NumElementi);
//* contiamo gli elementi diversi da 0
Cont=0;
```

```
                for (K=0;K<NumElementi;K++)
                {
                        if (Vettore[K] !=0 )
                        Cont++;
                }
}while (Cont!=1);

        //* L'elemento diverso da 0 del vettore (mcd) viene posto come primo elemento del vettore
        //* costruzione della matrice relativa alla trasformazione
        for (i=0;i<NumElementi;i++)     //* inizializzazione della matrice trasformazione come matrice identità
        {
                for (j=0;j<NumElementi;j++)
          {
                        if (i==j)
                                {
                                        Od [i][j]=1;
                                }
                        else
                        {
                                Od [i][j]=0;
                        }
          }
        }
        //* vanno scambiate le posizioni numero 1 e numero Pos del vettore se mcd non si trova in prima posizione
        if (Pos!=0){
                Od[Pos][Pos]=0;
                Od[0][0]=0;
                Od[Pos][0]=1;
                Od[0][Pos]=1;
```

```
            Moltiplica(Od);
}
/*
            controlliamo che il termine noto sia divisibile per mcd dei coefficienti e in caso affermativo vengono scritte le soluzioni
*/
if (Termine % MinVet==0){
                        //se è divisibile troviamo k e moltiplichiamo per k la prima riga della matrice U
                        intApp=Termine/MinVet;
                        if (intApp!=1){
                                    for (j=0;j<NumElementi;j++)
                                    {
                                                U[0][j]=intApp*U[0][j];
                                    }
                        }
                        // se una incognita dipende da un solo parametro negativo cambiamo i segni della riga
                        for (i=0;i<NumElementi;i++)     //* colonna della matrice U
                        {
                                    Cont=0;
                                    contM=0;
                                    Pos=0;
                                    for (j=1;j<NumElementi;j++)      //* riga della matrice U
                                    {
                                       if (U[j][i]!=0){
                                                      Cont++;
                                                      if (U[j][i]<0 )
                                                                  contM++;
                                                                  Pos=j;
                                                      }
                                    }
```

```
            if(contM==1 && Cont==1){
                    for (z=0;z<NumElementi;z++)
                            U[Pos][z]=-U[Pos][z];
            }
}
//scrittura delle soluzioni
for (i=0;i<NumElementi;i++)     //* colonna della matrice U
{
            strInd=intToString(i+1);
            str2 = "X"+strInd + "=";
            strApp="";
            if (U[0][i]!=0){
                    strApp=intToString(U[0][i]);
                    str2 = str2+strApp ;
            }
            for (j=1;j<NumElementi;j++)      //* riga della matrice U
            {
                    // costruzione della stringa
                    if (U[j][i]==0){
                            strApp="     ";
                             str2=str2+strApp;
                    }
                    else if (U[j][i]>0){
                            if (U[j][i]!=1){
                                    strApp=intToString(U[j][i]);
                            }
                            else {
                                    strApp="";
                            }
```

```
                                        str2=str2+" +"+ strApp;
                                        strInd=intToString(j);
                                        str2=str2 + "K" + strInd;
                                }
                                else {
                                        if (U[j][i]!=-1){
                                                strApp=intToString(U[j][i]);
                                        }
                                        else {
                                                strApp="-";
                                        }
                                        str2=str2+" " + strApp;
                                        strInd=intToString(j);
                                        str2=str2 + "K" + strInd;
                                }
                        }
                        cout <<str2;
                        cout << "\n";
                        cout << "\n";
                }
        }
        else {
                cout << "\n L'equazione non ammette soluzioni intere ";
        }
        return 0;
}
void Moltiplica ( int T[DimMax][DimMax])
{
        int Uappi [DimMax][DimMax];
```

```
int i, j, z;

 // moltiplicazione a sinistra della matrice U con la matrice Od

                for (i=0;i<NumElementi;i++)     //* riga  della matrice Od

                        {

                                for (j=0;j<NumElementi;j++)      //* colonna della matrice U

                                        {

                                                Uappi[i][j]=0;//*inizializzazione del valore della matrice

                                                for (z=0;z<NumElementi;z++)     //* colonna della matrice Od

                                                        {

                                                        Uappi[i][j]=Uappi[i][j]+(U[z][j]*T[i][z]);

                                                        }

                                        }

                        }

                for (i=0;i<NumElementi;i++)     //* i valori di Uapp vengono trasferiti su U

                                                {

                                                        for (j=0;j<NumElementi;j++)

                                                                {

                                                                        U[i][j]=Uappi[i][j];

                                                                }

                                                }

}
```