

Risoluzione di triangoli qualsiasi

```
*****
* risoluzione_triangoli_qualsiasi.mnb *
* ultima versione: 5 dicembre 2004 *
* testato con MuPAD Pro 3.1 su Windows XP sp2 *
*-----*
* Claudio Marsan *
* Liceo cantonale di Mendrisio *
* Via Agostino Maspoli *
* CH-6850 Mendrisio *
* claudio.marsan@liceomendrisio.ch *
*****
```

La procedura **risolviTriangolo** risolve un triangolo qualsiasi ABC dato in notazione standard:

vertici A, B, C in senso antiorario;

angoli α in A , β in B , γ in C dati in gradi sessadecimali;

lati a opposto ad α , b opposto a β , c opposto a γ .

La procedura richiede 6 argomenti: $a, b, c, \alpha, \beta, \gamma$. Per indicare che un elemento non è noto si scriverà "0" in corrispondenza del relativo argomento.

Gli argomenti devono avere forma numerica (per esempio: scrivere "sqrt(3.0)" invece di "sqrt(3)").

Devono essere dati esattamente tre argomenti diversi da "0", tra i quali almeno un lato.

- `reset()`;
- `DEG := PI/180: // "x*DEG" trasforma "x" in radianti`
- `RAD := 180/PI: // "x*RAD" trasforma "x" in gradi sessadecimali`

aux_lal risolve il problema parziale lato-angolo-lato

- ```
aux_lal := proc(L1, A3, L2)
 local L3, A2, A1;
 begin
 L3 := float(sqrt(L1^2 + L2^2 - 2*L1*L2*(cos(A3*DEG))));
 A2 := float(RAD * arccos((L1^2 + L3^2 - L2^2)/(2*L1*L3)));
 A1 := 180.0 - A2 - A3;
 return([L3, A1, A2]);
 end proc;
```

```

if temp = 1.0 then
 A2 := 90.0;
 A3 := 90.0 - A1;
 L3 := float(sqrt(L2^2 - L1^2));
 return([L3, A2, A3]);
end_if;
A21 := float(RAD * arcsin(temp));
A22 := 180.0 - A21;
A31 := 180.0 - A1 - A21;
A32 := 180.0 - A1 - A22;
sol := [];
if A31 > 0 then
 L31 := float(L1 * sin(A31*DEG)/sin(A1*DEG));
 sol := sol.[L31, A21, A31];
end_if;
if A32 > 0 then
 L32 := float(L1 * sin(A32*DEG)/sin(A1*DEG));
 sol := sol.[L32, A22, A32];
end_if;
return(sol);
end_proc:

```

### **aux\_laa risolve il problema parziale lato-angolo-angolo**

- `aux_laa := proc(L1, A1, A2)`  
`local A3, L2, L3;`  
`begin`  
`A3 := 180.0 - A1 - A2;`  
`L2 := float(L1 * sin(A2*DEG)/sin(A1*DEG));`  
`L3 := float(L1 * sin(A3*DEG)/sin(A1*DEG));`  
`return([L2, L3]);`  
`end_proc:`

### **La procedura principale**

- `risolviTriangolo := proc(a, b, c, alpha, beta, gamma)`  
`local num_lati, num_angoli, num_elementi, temp;`  
`begin`  
`/* Controllo dell'input */`  
`if (a < 0) or (b < 0) or (c < 0) or (alpha < 0) or (beta <`  
`0) or (gamma < 0) then`  
`return("ERRORE! Non sono ammesse misure negative.");`

```

if c <> 0 then
 num_lati := num_lati + 1;
end_if;
if alpha <> 0 then
 num_angoli := num_angoli + 1;
end_if;
if beta <> 0 then
 num_angoli := num_angoli + 1;
end_if;
if gamma <> 0 then
 num_angoli := num_angoli + 1;
end_if;
num_elementi := num_lati + num_angoli;
if (num_elementi <> 3) or (num_lati = 0) then
 return("ERRORE! Bisogna dare esattamente tre elementi, tra
i quali almeno un lato.");
end_if;

/* sono noti i tre lati: a, b, c */
if (num_lati = 3) then
 if (a >= b + c) or (b >= a + c) or (c >= a + b) then
 return("ERRORE! I tre lati non formano un triangolo.");
 end_if;
 alpha := float(RAD * arccos((b^2 + c^2 - a^2)/(2*b*c)));
 beta := float(RAD * arccos((a^2 + c^2 - b^2)/(2*a*c)));
 gamma := 180.0 - alpha - beta;
 return([a, b, c, alpha, beta, gamma]);
end_if;

/* sono noti due lati e l'angolo compreso: a, b, gamma */
if (a <> 0) and (b <> 0) and (gamma <> 0) then
 temp := aux_lal(a, gamma, b);
 return([a, b, temp[1], temp[2], temp[3], gamma]);
end_if;

/* sono noti due lati e l'angolo compreso: a, c, beta */
if (a <> 0) and (c <> 0) and (beta <> 0) then
 temp := aux_lal(c, beta, a);
 return([a, temp[1], c, temp[3], beta, temp[2]]);
end_if;

```

```

temp := aux_lla(a, b, alpha);
if nops(temp) = 3 then
 return([a, b, temp[1], alpha, temp[2], temp[3]]);
else
 return([a, b, temp[1], alpha, temp[2], temp[3], [a, b,
temp[4], alpha, temp[5], temp[6]]]);
end_if;
end_if;

```

```

/* sono noti due lati e un angolo non compreso: a, b, beta
*/
if (a <> 0) and (b <> 0) and (beta <> 0) then
 temp := aux_lla(b, a, beta);
 if nops(temp) = 3 then
 return([a, b, temp[1], temp[2], beta, temp[3]]);
 else
 return([a, b, temp[1], temp[2], beta, temp[3], [a, b,
temp[4], temp[5], beta, temp[6]]]);
 end_if;
end_if;

```

```

/* sono noti due lati e un angolo non compreso: a, c, alpha
*/
if (a <> 0) and (c <> 0) and (alpha <> 0) then
 temp := aux_lla(c, a, alpha);
 if nops(temp) = 3 then
 return([a, temp[1], c, alpha, temp[3], temp[2]]);
 else
 return([a, temp[1], c, alpha, temp[3], temp[2], [a,
temp[4], c, alpha, temp[6], temp[5]]]);
 end_if;
end_if;

```

```

/* sono noti due lati e un angolo non compreso: a, c, gamma
*/
if (a <> 0) and (c <> 0) and (gamma <> 0) then
 temp := aux_lla(c, a, gamma);
 if nops(temp) = 3 then
 return([a, temp[1], c, temp[2], temp[3], gamma]);
 else
 return([a, temp[1], c, temp[2], temp[3], gamma, [a,
temp[4], c, gamma, temp[6], temp[5]]]);
 end_if;
end_if;

```

```

 return([temp[1], b, c, temp[3], beta, temp[2]]);
 else
 return([temp[1], b, c, temp[3], beta, temp[2]],
[temp[4], b, c, temp[6], beta, temp[5]]);
 end_if;
end_if;

/* sono noti due lati e un angolo non compreso: b, c, gamma
*/
if (b <> 0) and (c <> 0) and (gamma <> 0) then
 temp := aux_lla(c, b, gamma);
 if nops(temp) = 3 then
 return([temp[1], b, c, temp[3], temp[2], gamma]);
 else
 return([temp[1], b, c, temp[3], temp[2], gamma],
[temp[4], b, c, temp[6], temp[5], gamma]);
 end_if;
end_if;

/* sono noti due angoli e un lato */
if (alpha = 0) and (beta <> 0) and (gamma <> 0) then
 alpha := 180.0 - beta - gamma;
end_if;
if (alpha <> 0) and (beta = 0) and (gamma <> 0) then
 beta := 180.0 - alpha - gamma;
end_if;
if (alpha <> 0) and (beta <> 0) and (gamma = 0) then
 gamma := 180.0 - alpha - beta;
end_if;
if (a <> 0) then
 temp := aux_laa(a, alpha, beta);
 return([a, temp[1], temp[2], alpha, beta, gamma]);
end_if;
if (b <> 0) then
 temp := aux_laa(b, beta, gamma);
 return([temp[2], b, temp[1], alpha, beta, gamma]);
end_if;
if (c <> 0) then
 temp := aux_laa(c, gamma, alpha);
 return([temp[1], temp[2], c, alpha, beta, gamma]);
end_if;

```

- `risolviTriangolo(50, 30, 0, 45, 0, 0)`  
[50, 30, 66.49012913, 45, 25.10409025, 109.8959097]
- `risolviTriangolo(0, 4.5, 9, 0, 0, 60)`  
[10.36249037, 4.5, 9, 94.34109373, 25.65890627, 60]
- `risolviTriangolo(12.4, 81.7, 0, 0, 36.7, 0)`  
[12.4, 81.7, 91.30523893, 5.20413364, 36.7, 138.0958664]
- `risolviTriangolo(3, 10, 0, 0, 0, 60)`  
[3, 10, 8.888194417, 16.99608806, 103.0039119, 60]
- `risolviTriangolo(200, 50, 177, 0, 0, 0)`  
[200, 50, 177, 110.4044027, 13.55117713, 56.04442022]
- `risolviTriangolo(0, 3.2, 2.4, 117, 0, 0)`  
[4.79304643, 3.2, 2.4, 117, 36.50308787, 26.49691213]
- `risolviTriangolo(20, 30, 0, 0, 45, 0)`  
[20, 30, 40.59964873, 28.1255057, 45, 106.8744943]
- `risolviTriangolo(5, 10, 0, 0, 0, 37.85)`  
[5, 10, 6.785131913, 26.88250874, 115.2674913, 37.85]
- `risolviTriangolo(15.7, 7.4, 9.6, 0, 0, 0)`  
[15.7, 7.4, 9.6, 134.4914133, 19.64742683, 25.86115983]
- `risolviTriangolo(74, 0, 0, 41, 77, 0)`

- `risolviTriangolo(0, 11.3, 6.7, 0, 52.20, 0)`  
`[14.08962083, 11.3, 6.7, 99.86309326, 52.2, 27.93690674]`
- `risolviTriangolo(82.6, 115.0, 0, 28.07, 0, 0) // due soluzioni`  
`[82.6, 115.0, 163.8790023, 28.07, 40.92905319, 111.0009468], [82.6, 115.0, 39.06687196, 28.07, 139.0709468, 12.85905319]`
- `risolviTriangolo(0, 127, 354.2, 0, 40.44, 0) // impossibile!`  
`Error: il triangolo non esiste! [aux_lla]`
- `risolviTriangolo(95.18, 92.20, 0, 0, 43.34, 0) // due soluzioni`  
`[95.18, 92.2, 134.2895223, 45.11373638, 43.34, 91.54626362], [95.18, 92.2, 4.158123363, 134.8862636, 43.34, 1.773736377]`
- `risolviTriangolo(2, 2.73, 0, 45, 0, 0)`  
`[2, 2.73, 2.453421589, 45, 74.84031866, 60.15968134], [2, 2.73, 1.407381437, 45, 105.1596813, 29.84031866]`
- `risolviTriangolo(5, 5, 5, 0, 0, 0) // triangolo equilatero`  
`[5, 5, 5, 60.0, 60.0, 60.0]`
- `risolviTriangolo(0, 0, 5, 60, 0, 60) // triangolo equilatero`  
`[5.0, 5.0, 5, 60, 60.0, 60]`
- `risolviTriangolo(2, 2, 0, 45, 0, 0) // triangolo rettangolo isoscele`  
`[2, 2, 2.828427125, 45, 45.0, 90.0]`
- `risolviTriangolo(0, 2, 0, 45, 45, 0) // triangolo rettangolo isoscele`

[38, 14.38782716, 45, 52.51552034, 17.48447966, 110]

- `risolviTriangolo(0, 8, 12, 0, 40, 0) // due soluzioni`

[11.31448261, 8, 12, 65.38143169, 40, 74.61856831], [7.070584027, 8, 12, 34.61856831, 40, 105.3814317]

- `risolviTriangolo(2*sqrt(3.0), 3*sqrt(3.0), 0, float(RAD*arcsin(2/3)), 0, 0) // in realtà: beta = 90`

[3.464101615, 5.196152423, 3.872983349, 41.8103149, 89.99999995, 48.18968515], [3.464101615, 5.196152423, 3.872983343, 41.8103149, 90.00000005]