Marco Milesi – 1030184 – Scala O.O.

# Realizzazione di un progetto SCALA per il calcolo degli zeri di funzione attraverso il metodo di Bisezione

INFORMATICA 3A  
Università degli Studi di Bergamo

**Introduzione**

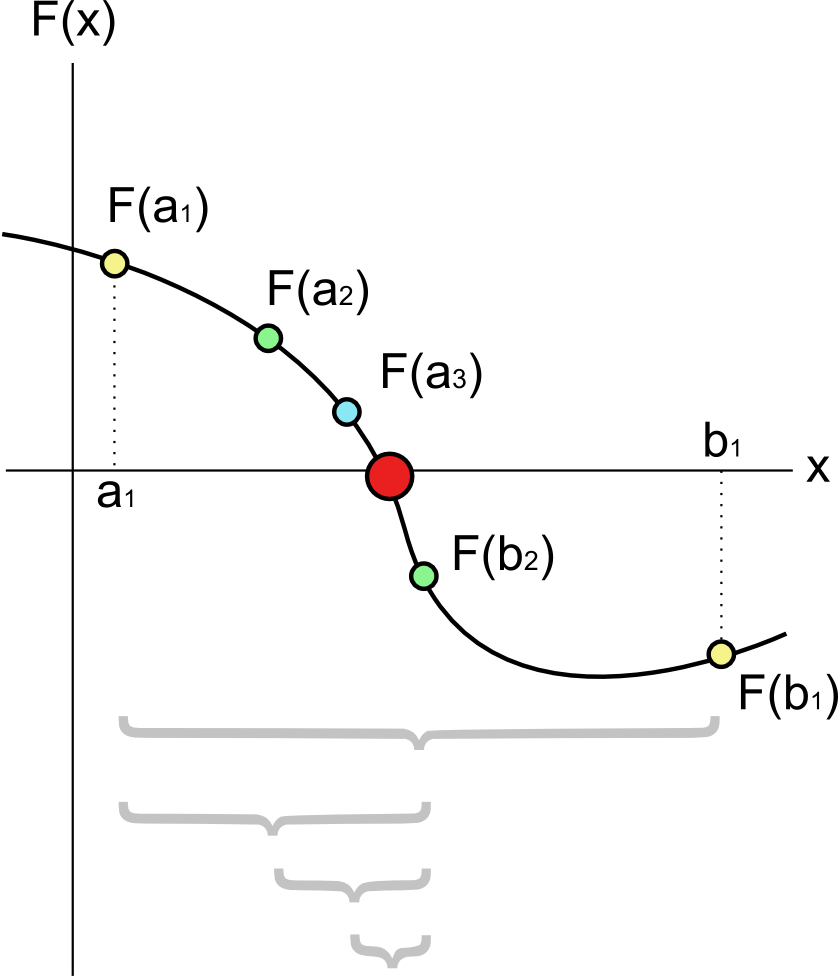
Il progetto è realizzato in linguaggio Scala con un’implementazione *Object Oriented* del metodo di bisezione e delle funzioni, nonché l’utilizzo di funzioni ricorsive per la risoluzione del problema.

Un metodo per approssimare le soluzioni di un'equazione di tipo:

eventualmente definita in un intervallo [a, b] è quello della bisezione, un algoritmo molto stabile in grado di riuscire a garantire un buon esito.

**Funzionamento**

Data un equazione **f(x) = 0** definita e continua in un intervallo [a,b] tale che f(a)\*f(b)<0 è possibile calcolarne un'approssimazione in [a,b], attraverso i seguenti passi.

* Separazione delle soluzioni attraverso l’individuazione di un intervallo [x1, x2] nel quale vi sia almeno una soluzione, ovvero nel quale ci sia un cambiamento di segno della funzione e quindi un attraversamento di f(x)=0.
* Si procede dividendo l'intervallo in due parti eguali e calcolando il valore della funzione nel punto medio di ascissa ***(a+b)/2***
* Se risulta f( (a+b)/2 ) = 0 allora la soluzione è la radice cercata.
* In caso contrario si individuano due nuovi intervalli [a, radice] e [radice, b] e si sceglie quello che assume valori di segno opposto agli estremi di funzione e si itera il procedimento (in modo **ricorsivo**) di dimezzamento ottenendo così una successione di intervalli, ognuno incluso nel precedente, fino ad arrivare alla soluzione o ad un’approssimazione della stessa in base al numero di iterazioni scelto (il programma consente la variazione di questo numero, impostato a 100 di default).

**Implementazione del Programma**

Il programma realizzato è composto di un singolo file, main.scala, che racchiude un oggetto “main” e due classi:

* **Func:** rappresenta l’oggetto che contiene la funzione. Nel programma, a titolo di esempio, sono utilizzati 3 tipi di funzione.

// y = 5x + 3 : -0.6

**var** Retta = **new** Func(List(5, 3));

// y = 1\*x^2-2\*x^0 : 1.44

**var** Parabola = **new** Func(List(1, 0, -2));

// y = 2 x^3 - 2 x^0 : 1

**var** Cubica = **new** Func(List(2, 0, 0, -2))

// y = 2 x^3 - 1 x^0 : 0.79

**var** Cubica2 = **new** Func(List(2, 0, 0, -1))

L’oggetto Func è quindi composto da una lista di interi, e presenta un metodo ***ValutaFunc(x: Double) = { … }*** che, ad un input x in entrata, computa il suo corrispettivo f(x) attraverso il seguente ciclo for:

**for**(i <- 0 until **this**.params.size) {

r += (**this**.params.apply(i) \* pow(x, **this**.params.size - i - 1));

}

Per esempio, la computazione eseguita nel ciclo for per una List(1,0,-2) effettua la seguente operazione:

r = ( f(x) ) = 1 \* x ^ 2 + 0 \* x ^ 1 – 2 \* x ^ 0

* **Bisection:** rappresenta l’oggetto che implementa l’algoritmo della bisezione e contiene due metodi:
  + **Valuta::Double()**: ritorna il risultato del metodo di bisezione, a partire dall’oggetto definito nella classe Func (la funzione), gli estremi dell’intervallo di controllo del metodo e il valore intero delle iterazioni configurato in una variabile finale val impostata a 100 di default.
  + **ValutaToString::String():** ritorna la valutazione di Valuta::Double convertendo il risultato in una stringa con livello di precisione impostato nella variabile finale val, impostata a 5 di default.

Per lo svolgimento di alcuni calcoli viene inoltre utilizzata la libreria Math, di cui il programma carica e richiama il metodo pow(a, b) che esegue l’operazione di potenza a^b.

**Configurazioni**

Il programma presenta due variabili utilizzabili per configurarne il comportamento rispetto al numero di iterazioni e alla precisione da utilizzare per il ritorno del risultato in stringa.

**class** Bisection() {

// Precision per la funzione ToString

**val** precision = 5;

...

}

**object** main {

// Numero massimo di iterazione da computare

**val** iterazioni = 100;

...

}

**Codice del progetto**

**import** Math.pow

**object** main {

// Numero massimo di iterazione da computare

**val** iterazioni = 100;

**def** main(args: Array[String]): Unit = {

// y = 5x + 3 : -0.6

**var** Retta = **new** Func(List(5, 3));

// y = 1\*x^2-2\*x^0 : 1.44

**var** Parabola = **new** Func(List(1, 0, -2));

// y = 2 x^3 - 2 x^0 : 1

**var** Cubica = **new** Func(List(2, 0, 0, -2));

// y = 2 x^3 - 1 x^0 : 0.79

**var** Cubica2 = **new** Func(List(2, 0, 0, -1));

**var** Bisection = **new** Bisection();

println( Bisection.Valuta(Retta)(2)(-2)(iterazioni) );

println( Bisection.Valuta(Parabola)(2)(-2)(iterazioni) );

println( Bisection.ValutaToString(Parabola)(2)(-2)(iterazioni) );

println( Bisection.Valuta(Cubica)(2)(-2)(iterazioni) );

println( Bisection.Valuta(Cubica2)(2)(-2)(iterazioni) );

}

}

**class** Func(**var** l: List[Double]) {

**var** params = l;

// Valutatore di funzione in un punto x: Double

**def** ValutaFunc(x: Double) = {

**var** r: Double = 0;

**for**(i <- 0 until **this**.params.size) {

r += (**this**.params.apply(i) \* pow(x, **this**.params.size - i - 1));

}

r;

}

}

**class** Bisection() {

// Precision per la funzione ToString

**val** precision = 5;

**def** Valuta(f: Func)(s: Double)(e: Double)(iterazioni: Int): Double = {

**var** root = (s + e) / 2;

**var** valore: Double = f.ValutaFunc( root );

**if** (valore == 0 || iterazioni == 0) {

root;

} **else** {

**if** (valore > 0) {

**if** (s < e) {

**this**.Valuta(f)(s)(root)(iterazioni - 1);

} **else** {

**this**.Valuta(f)(root)(e)(iterazioni - 1);

}

} **else** {

**if** (s < e) {

**this**.Valuta(f)(root)(e)(iterazioni - 1);

} **else** {

**this**.Valuta(f)(s)(root)(iterazioni - 1);

}

}

}

}

**def** ValutaToString(f: Func)(s: Double)(e: Double)(iterazioni: Int): String = {

String.valueOf(**this**.Valuta(f)(s)(e)(iterazioni)).slice(0,precision);

}

}