

Equazioni Non Lineari

Metodi iterativi pratici per l'approssimazione di radici

Leonardo Volpi

INDICE

INTRODUZIONE	3
ZERI REALI DI FUNZIONE REALE.....	4
CONCETTI E DEFINIZIONI	4
<i>Radici reali e interpretazione geometrica.....</i>	4
<i>Separazione delle radici (bracketing).....</i>	4
<i>Soluzione simbolica e numerica.....</i>	6
<i>Zero numerico.....</i>	8
<i>Radici multiple.....</i>	8
<i>Il fenomeno del "breakdown".....</i>	9
<i>Criteri di arresto.....</i>	9
<i>Il fenomeno del "bouncing".....</i>	11
VELOCITÀ DI CONVERGENZA	13
<i>Ordine e costante di convergenza.....</i>	13
<i>Stima dell'ordine di convergenza.....</i>	15
<i>Traiettorie di convergenza.....</i>	17
<i>Stima statistica dell'ordine di convergenza.....</i>	18
ALGORITMI.....	20
<i>Flow-chart.....</i>	20
<i>Costo computazionale.....</i>	21
<i>Calcolo delle Derivate.....</i>	23
<i>Metodo del Punto Fisso.....</i>	24
<i>Accelerazione di Aitken.....</i>	26
<i>Metodo di Bisezione.....</i>	28
<i>Metodo Regula Falsi.....</i>	31
<i>Metodo della Secante.....</i>	33
<i>Metodo Secante back-step.....</i>	35
<i>Metodo Pegasus.....</i>	36
<i>Metodo Newton-Raphson.....</i>	38
<i>Metodo Halley.....</i>	43
<i>Metodo Halley con differenze finite.....</i>	45
<i>Metodo della Parabola.....</i>	46
<i>Metodo della Parabola Inversa.....</i>	49
<i>Metodo della Frazione Lineare.....</i>	51
<i>Metodo Muller.....</i>	54
<i>Metodo Star.....</i>	55
<i>Metodo Chebyshev.....</i>	58
<i>Metodo Chebyshev con differenze finite (Secantx).....</i>	60
<i>Metodo Wijngaardern- Dekker-Brent.....</i>	61
<i>Metodo Rheinboldt (2).....</i>	63
<i>Metodo Steffensen.....</i>	64
<i>Formula Householder.....</i>	65
<i>Formula di Povposki.....</i>	66
TEST	68
<i>Casi di Test.....</i>	68
<i>Costo per iterazione.....</i>	72
<i>Classifica Affidabilità.....</i>	72
<i>Classifica Iterazioni.....</i>	73
<i>Classifica Costo.....</i>	74
<i>Classifica Efficienza.....</i>	74
ZERI DI POLINOMI.....	76
<i>Proprietà dei polinomi.....</i>	76
<i>Divisione veloce di polinomi.....</i>	77
<i>Calcolo del polinomio e delle sue derivate.....</i>	79
<i>Calcolo di polinomi in precisione finita.....</i>	81

<i>Effetto barriera</i>	82
<i>Deflazione</i>	84
<i>Deflazione con decimali</i>	84
<i>Deflazione reciproca</i>	86
<i>Traslazione</i>	87
RICERCA DELL'AREA DELLE RADICI	90
<i>Metodo grafico</i>	90
<i>Metodo della matrice compagna</i>	91
<i>Metodo del cerchio delle radici</i>	92
<i>Metodo iterativo delle potenze</i>	94
<i>Metodo quoziente-differenza (QD)</i>	95
RICERCA DELLE RADICI MULTIPLE	101
<i>Massimo comun divisore</i>	101
<i>Massimo comun divisore con decimali</i>	102
<i>MCD - Algoritmo pratico di Euclide</i>	106
<i>MCD - Test dei residui dei cofattori</i>	107
<i>MCD - La matrice di Sylvester</i>	108
<i>Metodo delle derivate</i>	109
<i>Stima della molteplicità</i>	110
RICERCA DELLE RADICI INTERE	113
<i>Tentativi successivi</i>	113
<i>Tentativi per intervalli</i>	115
ALGORITMI	116
<i>Metodo Newton-Raphson complesso</i>	116
<i>Metodo Halley complesso</i>	120
<i>Metodo Lin-Bairstow</i>	122
<i>Metodo Siljak</i>	128
<i>Metodo Laguerre</i>	129
<i>Metodo a convergenza simultanea (ADK)</i>	131
<i>Zeri dei polinomi ortogonali</i>	134
<i>Metodo Jenkins-Traub</i>	139
<i>Metodo QR</i>	142
TEST PER POLINOMI	145
<i>Test Random</i>	145
<i>Test Wilkinson</i>	146
<i>Test di selettività</i>	147
STRATEGIA DI RICERCA DELLE RADICI DEI POLINOMI	148
<i>Strategia d'attacco</i>	148
<i>Esempi di risoluzione di polinomi</i>	148
BIBLIOGRAFIA	152
ALLEGATI	154
RISULTATI DEI CASI DI TEST	154
<i>Costo per metodo e test</i>	154
<i>Numero d'iterazioni per metodo e test</i>	154
POTENZE DEL BINOMIO COMPLESSO (X+IY)	160
SOFTWARE	161

Introduzione

Le radici di un'equazione non lineare possono essere espresse esplicitamente in forma analitica chiusa molto raramente. Anche se ciò fosse possibile spesso l'espressione è talmente laboriosa e complicata da essere praticamente inutilizzabile dal punto di vista del calcolo numerico. Pertanto per risolvere numericamente equazioni non lineari si ricorre a metodi approssimati di tipo iterativo (rootfinder); partendo da una stima iniziale, essi producono, passo dopo passo, una successione di soluzioni sempre più vicine a quella effettiva. Di questi metodi ne sono stati sviluppati nel corso del tempo moltissimi, alcuni veramente geniali.

In questo documento vengono presentati e analizzati necessariamente solo alcuni di questi algoritmi, orientati al calcolo scientifico, scelti in base alle loro peculiarità di efficienza, velocità, e semplicità. L'avvento di calcolatori ha in parte modificato l'approccio verso questi metodi. Nel calcolo manuale venivano preferiti un tempo gli algoritmi semplici ma al tempo stesso ad alta velocità di convergenza che sfruttavano informazioni sulla funzione e le sue derivate. Con il calcolo automatico si tende a prediligere metodi più lenti ma più robusti con alta garanzia di successo anche nella più disparate situazioni e in assenza di controllo umano.

Vedremo che la scelta di un metodo dipenderà da diversi fattori, quali la disponibilità di una stima iniziale accurata della soluzione, il costo del calcolo della funzione e delle sue derivate, la possibilità di avere la funzione e le derivate in forma analitica esatta, il loro comportamento, l'accuratezza richiesta, ecc.

Non esiste quindi un metodo che possa essere definito migliore di tutti ma dipenderà dal tipo di applicazione e dal problema considerato.

Questo documento è essenzialmente pratico, cioè rivolto a chi, per lavoro o per studio, è alla ricerca di un buon metodo che funzioni ragionevolmente bene e che sia semplice da programmare.

Numerosi esempi forniranno le spiegazioni senza ricorrere a teoremi e dimostrazioni.

Zeri reali di funzione reale

Concetti e Definizioni

In questo capitolo ci occuperemo della ricerca delle radici reali di equazioni non lineari. Data una funzione $f(x)$ definita nel campo dei numeri reali, vogliamo trovare i valori x reali che soddisfano all'equazione:

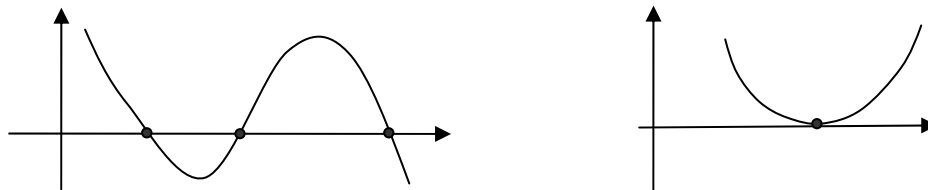
$$f(x) = 0.$$

Tali valori si dicono "radici" dell'equazione.

Radici reali e interpretazione geometrica

Geometricamente parlando le radici reali dell'equazione sono i punti d'incontro della funzione con l'asse delle ascisse, per cui sono anche detti "zeri" della funzione $f(x)$.

Va detto che la funzione può avere anche degli zeri che non attraversano l'asse x . Tali punti sono detti stazionari o estremi e, in generale, non vengono trattati con i metodi descritti in questi documenti, ma con i metodi di minimizzazione delle funzioni.

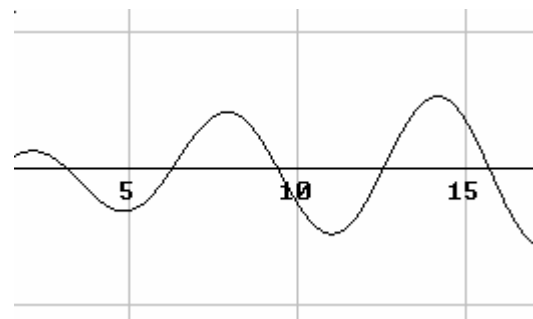


Il nostro studio si occuperà della ricerca ed approssimazione dei punti di attraversamento dell'asse x

Separazione delle radici (bracketing)

La maggior parte dei metodi numerici per la ricerca degli zeri di una funzione $f(x)$ (ovvero le soluzioni o radici dell'equazione $f(x) = 0$) richiedono prima di tutto la ricerca degli intervalli nei quali si trova un solo zero; questa operazione è detta di separazione delle radici o "bracketing".

Il metodo più importante, consiste nello studiare la funzione $f(x)$, disegnare il grafico, e ricavare gli intervalli in cui cadono gli zeri. Le tecniche a cui si ricorre sono quelle dell'analisi classica. Con l'aiuto di un programma matematico di grafica e uno spreadsheet si può ottenere velocemente la tabulazione e il grafico di una funzione reale comunque complicata.



Tali programmi (ve ne sono di eccellenti anche freeware) sono ormai alla portata di tutti. Mediante adatti tools quali "zoom" and "shift" si può arrivare ad isolare le singole radici ed approssimarle con la precisione migliore del 1%. Per precisioni maggiori il metodo tende a diventare tedioso e si preferisce ricorrere ai metodi numerici di approssimazione iterativi partendo dalle stime iniziali fornite dallo studio del grafico. L'osservazione del comportamento della funzione nell'intorno delle zero ci permetterà anche di scegliere l'algoritmo iterativo più adatto.

Da quanto detto emerge che la separazione delle radici deve partire sempre da uno studio analitico manuale della funzione, aiutato, al più, da opportuni "tools" di grafica matematica. Non si conoscono programmi completamente automatici per la separazione delle radici delle equazione generale $f(x) = 0$ (ad eccezione forse dei polinomi, come vedremo)

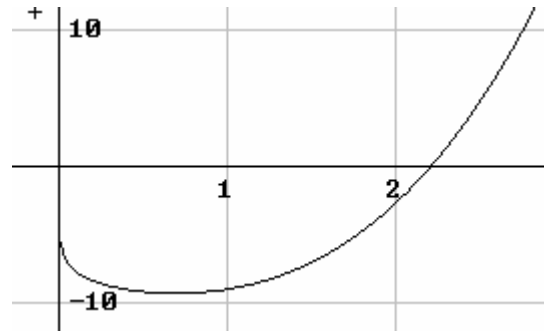
Quanto sia importante lo studio analitico di una funzione per la ricerca degli zeri lo dimostra il seguente semplice esempio

Esempio 1. Trovare gli zeri della funzione: $f(x) = x^3 - \log(x) - 10$

Tracciamo il grafico nell'intervallo $0 < x < 3$ campionando la funzione con un passo molto piccolo, ad esempio $h = 0.01$.

In genere questo è il tipo di elaborazione che eseguono i cosiddetti programmi di "ricerca automatica" delle radici.

Con circa 300 punti la funzione cambia di segno solamente fra 2.0 e 2.2



Se ripetiamo con 3000 punti ($h = 0.001$), e quindi con un costo computazionale molto alto, non otteniamo nessun'altra informazione aggiuntiva a parte una riduzione dell'intervallo della radice intorno a 2.

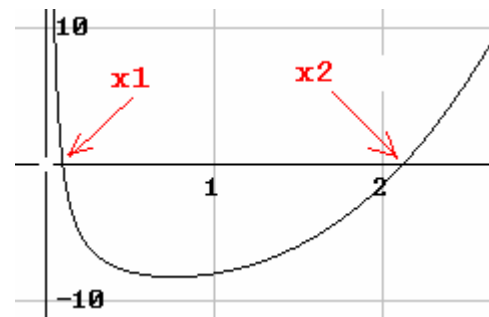
Dovremmo concludere quindi che esiste un solo zero della funzione nell'intervallo $[0, 3]$. Ma sbaglieremmo, perché una semplice analisi rivela la presenza di almeno due radici.

Infatti prendendo i limiti

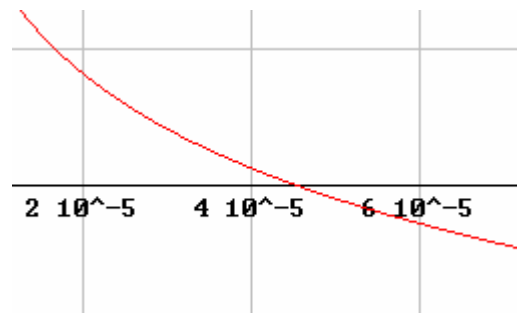
$$\lim_{x \rightarrow 0^+} f(x) = \lim_{x \rightarrow 0^+} (x^3 - \log(x) - 10) = +\infty$$

$$\lim_{x \rightarrow +\infty} f(x) = \lim_{x \rightarrow +\infty} (x^3 - \log(x) - 10) = +\infty$$

Inoltre essendo $f(1) = -9$ (negativo) devono quindi esistere almeno due zeri - x_1 e x_2 - localizzati rispettivamente a sinistra e a destra del punto 1



La radice x_1 è molto difficile da individuare; ma poiché sappiamo che "deve" esistere concentriamo la nostra ricerca solo attorno l'origine con passo più piccolo ($h = 1E-5$)



In effetti, ora, mediante lo "zooming" riusciamo ad isolare la radice "nascosta" in un segmento $[4E-5, 6E-5]$.

Soluzione simbolica e numerica

Le radici di un'equazione non lineare non possono in generale essere trovate in forma simbolica esatta e si deve ricorrere ai metodi di approssimazione numerica. Se pur semplice, il concetto di "esatto" può creare qualche confusione. Chiariamo pertanto che cosa s'intende per soluzione "esatta" e soluzione numerica.

Una radice di un'equazione è in forma simbolica esatta se è possibile esplicitare la variabile x attraverso un'espressione chiusa contenente numeri e/o simboli ed eventualmente anche altre funzioni matematiche. In pratica deve essere scritta in una forma del tipo

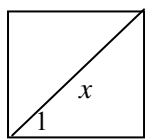
$$x = \left\{ 1 + \frac{2}{3}a^2 + g(a, b, \dots) + \dots \right\}$$

Dove l'espressione a sinistra del segno "=" non contiene la variabile x , e deve essere chiusa, cioè per calcolarla si deve eseguire un numero finito di operazioni.

L'espressione simbolica esatta o analitica è la forma più generale possibile della soluzione. Ma conoscere la radice simbolica esatta non significa automaticamente avere la soluzione numerica esatta, perché può non essere rappresentabile esattamente con un numero finito di cifre.

Vediamo qualche esempio scelti fra quelli più familiari

$x = 2 \cdot \pi$ $x = 3.14\dots$	Circonferenza di raggio unitario: si conosce la soluzione simbolica esatta ma solo una soluzione numerica approssimata
--------------------------------------	--



$$x = \sqrt{2}$$

$$x = 1.41\dots$$

Diagonale del quadrato di lato unitario: conosciamo la soluzione simbolica esatta ma solo una soluzione numerica approssimata

$4x^2 - 9 = 0$ $x_{1,2} = \pm \frac{3}{2}, x_{1,2} = \pm 1.5$	Dell'equazione algebrica a sinistra conosciamo sia la soluzione simbolica esatta che la soluzione numerica esatta.
--	--

$x^3 - 2 = 0$ $x = \sqrt[3]{2}, x = 1.26\dots$	Dell'equazione algebrica a sinistra conosciamo la soluzione simbolica esatta ma solo una soluzione numerica approssimata
---	--

$\tan(x) - 1 = 0$ $x = \frac{\pi}{4}, x = 0.785\dots$	Dell'equazione trascendente a sinistra conosciamo la soluzione simbolica esatta ma solo una soluzione numerica approssimata
--	---

$e^{5x} - e^7 = 0$ $x = \frac{7}{5}, x = 1.4$	Dell'equazione trascendente a sinistra conosciamo sia la soluzione simbolica esatta che la soluzione numerica esatta.
--	---

$2 \cdot \log(x) - 1 = 0$ $x = e^{0.5}, x = 1.64\dots$	Dell'equazione trascendente a sinistra conosciamo la soluzione simbolica esatta ma solo una soluzione numerica approssimata
---	---

Molto spesso non conosciamo neppure la soluzione simbolica esatta. Oppure la soluzione simbolica è nota ma troppo complicata.

$$\cos(x) - x = 0$$

Dell'equazione trascendente a sinistra non conosciamo la soluzione simbolica esatta

$$27x^3 + 18x = -128$$

$$x = \sqrt[3]{\frac{2\sqrt{114}}{9} - \frac{64}{27}} - \sqrt[3]{\frac{2\sqrt{114}}{9} + \frac{64}{27}}$$

Dell'equazione algebrica a sinistra conosciamo la soluzione simbolica esatta ma è poco utile perchè molto complicata sia dal punto di vista simbolico che da quello numerico

Negli ultimi due casi ricorriamo ai metodi numerici iterativi.

Zero numerico

Come abbiamo visto la ricerca delle radice dell'equazione $f(x) = 0$ consiste nel trovare quei punti in cui la funzione stessa diventa nulla. Un noto teorema conosciuto come "esistenza dello zero" afferma che:

Se una funzione $f : \mathbb{R} \rightarrow \mathbb{R}$, continua, definita in un intervallo $[a, b]$ per cui si verifica $f(a) \cdot f(b) < 0$, allora esiste senz'altro almeno un punto interno $c \in (a, b)$ per cui $f(c) = 0$

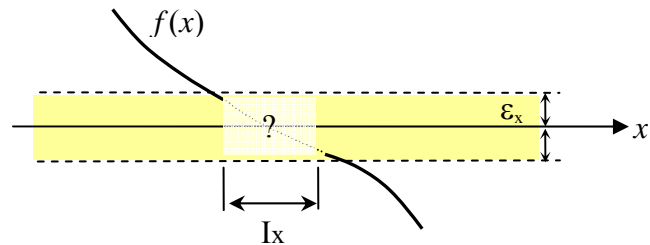
Tuttavia dal punto di vista numerico la verifica dello zero non è banale. La precisione di macchina finita e gli errori di arrotondamento limitano la possibilità di raggiungere lo zero ideale. Pertanto dobbiamo definire meglio cosa s'intende per "zero numerico". Per questo sostituiamo il simbolo "=" con quello "≈" e definiamo.

$$f(x) = 0 \Rightarrow f(x) \approx 0 \Rightarrow \exists I_x : |f(x)| < \epsilon_m \quad \forall x \in I_x$$

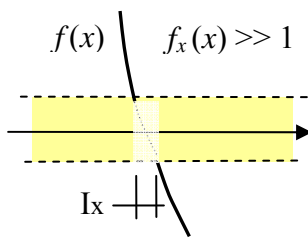
Quindi diciamo che la funzione ha raggiunto lo zero numerico se esiste un intervallo I_x tale che, per ogni x di questo intervallo, i valori della funzione sono tutti minori di un prefissato numero ϵ_m , piccolo (ma non a piacere); in genere si fissa la precisione di macchina.

Dal punto di vista concettuale, ogni punto dell'intervallo I_x sarebbe uno zero numerico della funzione; per comodità si conviene di assumere il punto medio dell'intervallo come zero numerico.

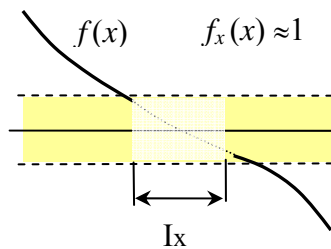
In pratica i metodi numerici iterativi sono programmati per interrompere il processo



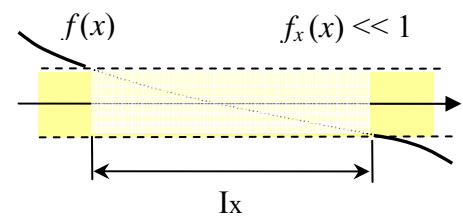
non appena la funzione cade nella fascia di errore di macchina $\pm \epsilon_m$. La larghezza dell'intervallo I_x e quindi anche l'accuratezza della radice, dipendono dalla pendenza locale della funzione $f(x)$. Possono cioè presentarsi i seguenti casi.



L'intervallo della radice è molto minore di quello della precisione di macchina. Accuratezza della radice molto alta



L'intervallo della radice è paragonabile a quello della precisione di macchina. Accuratezza della radice alta



L'intervallo della radice è molto maggiore di quello della precisione di macchina. Accuratezza della radice molto bassa

Radici multiple.

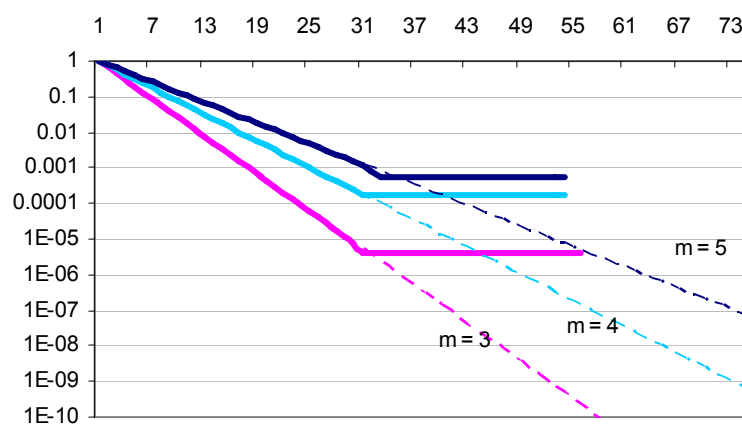
Il terzo caso è sicuramente quello più critico (ill-conditioned problem) perché l'approssimazione della radice è molto più bassa di quella ottenibile negli altri due casi. Di solito si verifica in presenza di radici multiple. In tali casi anche la derivata della funzione tende a valori molto piccoli e di conseguenza la tangente alla curva diventa parallela all'asse x . Anche se la funzioni ha raggiunto i valori piccolissimi del limite di macchina, l'intervallo contenente la radice rimane di diversi ordini di grandezza più grande. Ogni iterazione in più non può più restringere ulteriormente l'intervallo. Si deve accettare, in questi casi, una precisione minore per la radice trovata.

Il fenomeno del "breakdown"

Gli esempi seguenti, relativi a polinomi di 3, 4, e 5 grado mostrano l'andamento della precisione in presenza di radici multiple. Anche la generica funzione $f(x)$ ha un comportamento simile, perché ogni funzione continua e derivabile può essere sempre approssimata con il polinomio di Taylor con precisione comunque arbitraria.

Polinomio	Radice	Moltep.	Precisione
x^3-3x^2+3x-1	1	3	$10^{-5} - 10^{-6}$
$x^4-4x^3+6x^2-4x+1$	1	4	$10^{-4} - 10^{-5}$
$x^5-5x^4+10x^3-10x^2+5x-1$	1	5	$10^{-3} - 10^{-4}$

Il grafico sotto riporta l'andamento dell'errore in funzione delle iterazioni di un metodo numerico (il metodo di Newton in questo caso) su PC a 32 bit, con precisione di macchina $\epsilon_m \approx 1E-16$



Si nota il caratteristico fenomeno di "barriera" oltre il quale l'errore non può più scendere. Ogni ulteriore iterazione è inutile e il processo si deve fermare. Il livello della barriera è in funzione della molteplicità della radice e della precisione di macchina. Dal grafico si vede che se una radice ha molteplicità 3, la precisione massima è di circa $1E-5$, vale a dire circa 5-6 cifre significative massime.

Criteri di arresto

Al fine di garantire l'arresto del processo iterativo i programmi automatici devono esaminare ad ogni passo se esistono le condizioni per proseguire. Questi test prendono il nome di criteri di arresto (stopping tests) e possono essere più o meno elaborati in funzione del problema e delle condizioni in cui essi operano. In generale però i metodi iterativi ricadono in due famiglie:

Quelli che generano una sequenza di valori: $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ convergenti alla radice (fixed-point methods) o quelli che generano una successione d'intervalli $[a_i, b_i]$ contenenti la radice (bracketing methods). Per tutte e due le famiglie i comuni criteri di arresto sono i seguenti

- Ampiezza del residuo della funzione: $|f(x)| < \epsilon$;
- Ampiezza dell'intervallo: $|a_i - b_i| < \epsilon$; (solo per successione d'intervalli)
- Ampiezza dell'incremento: $|x_{n-1} - x_n| < \epsilon$;
- Massimo numero d'iterazioni: $IT < IT_{max}$;

Oltre a questi criteri che possiamo definire "assoluti", vi sono anche i corrispondenti criteri "relativi", che controllano l'ampiezza relativa dell'intervallo o dell'incremento. Per evitare

l'overflow, questi criteri si applicano però quando i valori medio dell'intervallo o dell'incremento sono maggiori di 1.

- Ampiezza relativa del intervallo: $|a_i - b_i| < \varepsilon |a_i + b_i|$;
- Ampiezza relativa del incremento: $|x_{n-1} - x_n| < \varepsilon |x_{n-1} + x_n|$;

I criteri relativi sono necessari per evitare che un processo iterativo cada in un ciclo infinito (loop) Ad esempio se una radice è dell'ordine di $x_n \approx 1E+8$, il più piccolo incremento che possiamo avere con una precisione di macchina di $\varepsilon = 1E-16$ è dell'ordine di $|x_{n-1} - x_n| \approx 1E-6$, che è molto maggiore di ε . In questo caso il criterio assoluto di arresto non funzionerebbe e il processo iterativo non terminerebbe mai. Al contrario, quello relativo intercetta subito la condizione di limite raggiungibile. In pratica si adottano quasi sempre entrambi i criteri (criterio misto). Quando i valori medi sono maggiori di 1 si adotta il criterio relativo, altrimenti si adotta quello assoluto.

Va subito detto che dal punto di vista della sicurezza di funzionamento è molto meglio che un processo termini prematuramente piuttosto che cada in un loop. Quindi molto spesso, nei programmi reali, i limiti di arresto sono piuttosto cautelativi

Oltre ai sopradescritti criteri di arresto, per particolari situazioni si adottano anche altri test più sofisticati, che si basano su metodi statistici, quali la media mobile e la regressione.

In pratica alle successioni dei valori o della radice $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ o della funzione $f_0, f_1, f_2, \dots, f_n$ si sostituiscono le loro medie mobili su un certo numero di valori. Ad esempio per 4 valori si hanno le seguenti successioni delle medie

$$\tilde{x}_i = \frac{x_i + x_{i-1} + x_{i-2} + x_{i-3}}{4} \quad \tilde{f}_i = \frac{f_i + f_{i-1} + f_{i-2} + f_{i-3}}{4}$$

I test di arresto vengo quindi fatti sulle successioni delle medie anziché sui valori puntuali. Questo criterio diminuisce le probabilità di errate decisioni quando l'algoritmo iterativo si trova ad operare vicino al limite di macchina.

Il fenomeno del "bouncing"

Operando con aritmetica in virgola mobile a precisione finita anche un altro curioso fenomeno può entrare in gioco quando ci si avvicina al limite di precisione macchina. Tale fenomeno è evidente soprattutto nel calcolo dei polinomi di alto grado.

Quando la successione dei valori $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ si avvicina alla radice i corrispondenti valori del polinomio $P_0, P_1, P_2, \dots, P_n$ iniziano a "rimbalzare" caoticamente in modo sempre più accentuato man mano che ci si avvicina al limite di macchina. La traiettoria dell'errore diviene casuale e i comuni criteri di stop possono fallire. In questa situazione infatti gli ultimi due valori della successione non sono necessariamente i più accurati e il test può portare a decisioni sbagliate. Ad esempio il seguente polinomio di 10° grado ha la radice $x = 10$:

$$55064880 - 86622048x + 62847156x^2 - 27612760x^3 + 8104060x^4 - 1652944x^5 + 236293x^6 - 23290x^7 + 1510x^8 - 58x^9 + x^{10}$$

Effettuiamo il calcolo del polinomio con il metodo di Ruffini-Horner che è il più comune e il più efficiente fra i metodi di calcolo numerico dei polinomi

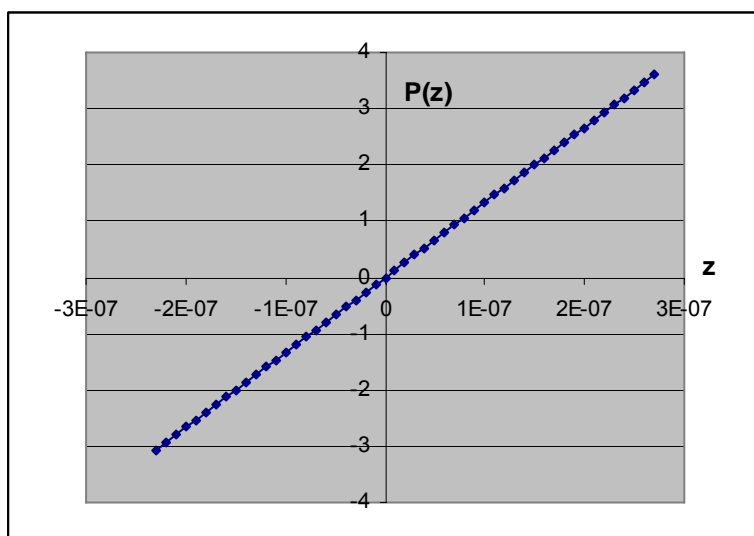
$$p = 0, \quad p = x \cdot p + \text{coefficiente}(i), \quad i = 10, 9 \dots 0$$

Il metodo può essere impostato in forma tabellare in un comune spreadsheet

	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
	1	-58	1510	-23290	236293	-1652944	8104060	-27612760	62847156	-86622048	55064880
10	0	10	-480	10300	-129900	1063930	-5890140	22139200	-54735600	81115560	-55064880
	1	-48	1030	-12990	106393	-589014	2213920	-5473560	8111556	-5506488	0

Inserendo il valore di x nella prima cella a sinistra (gialla) otteniamo il corrispondente valore $f(x)$ nell'ultima cella a destra (grigia). I valori della prima riga sono i coefficienti del polinomio ordinati da sinistra a destra secondo le potenze decrescenti. I valori dell'ultima riga sono la somma in verticale delle due celle sovrastanti. I valori della riga intermedia, dalla seconda colonna in poi, sono il prodotto della x per il valore della terza riga della colonna precedente.

Ora immaginiamo di essere un programma di ricerca di radici e iniziamo a tabulare il polinomio nell'intervallo attorno alla radice $[10 - e, 10 + e]$ e riportiamo in un grafico i valori del polinomio



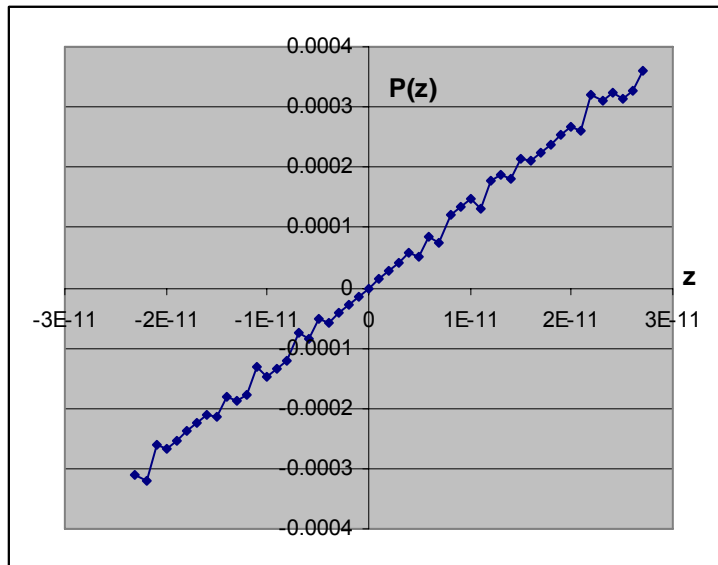
Intervallo $[10 - e, 10 + e]$
con $e = 3E-7$

Per comodità la variabile x è stata normalizzata.

$$z = (x - 10) / 10$$

i punti blu sono i corrispondenti valori del polinomio $P(z)$
La convergenza a zero è lineare e regolare.

Continuiamo la discesa prendendo un intervallo più stretto. Ad esempio $e = 3E-11$

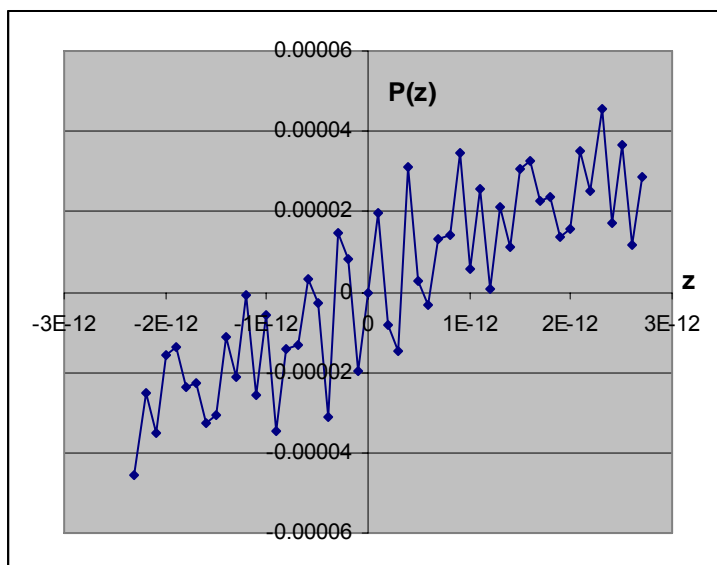


Intervallo $[10 - \epsilon, 10 + \epsilon]$
con $\epsilon = 3E-11$

La convergenza è ancora evidente ma iniziano ad apparire strane perturbazioni apparentemente casuali. Questo è un segno che siamo ormai prossimi al limite di accuratezza

Notare che i valori del polinomio sono ancora relativamente alti (circa $3E-4$) mentre l'accuratezza della radice è dell'ordine di $3E-11$

Se proviamo a restringere ulteriormente l'intervallo, il fenomeno delle perturbazioni casuali (bouncing) diviene ancor più evidente



Intervallo $[10 - \epsilon, 10 + \epsilon]$
dove $\epsilon = 3E-12$

Il fenomeno del "bouncing" ci segnala che siamo ormai prossimi al limite della massima accuratezza possibile per questo polinomio. Un algoritmo iterativo non può più andare oltre.

Notare che i valori del polinomio sono ancora relativamente alti (circa $3E-4$) mentre l'accuratezza della radice è dell'ordine di $3E-11$

Si nota inoltre che la precisione della radice in questo caso è di diversi ordini maggiore delle precisione di macchina ($1E-16$).

Nell'intervallo minimo raggiunto il calcolo del polinomio è praticamente casuale. Si notano dei valori del polinomio prossimi a zero per x diversi dall'origine e, per contro, valori di x molto prossimi all'origine possono ritornare dei valori relativamente alti del polinomio.

Chiaramente ogni decisione presa in questa situazione per mezzo di semplici test quali

$$|f(x_n)| < \epsilon \quad \text{oppure} \quad |x_{n-1} - x_n| < \epsilon$$

può facilmente portare a decisioni errate.

L'approccio statistico - ad esempio la media mobile degli ultimi 8 valori - può in questo caso portare ad una riduzione dell'intervallo della radice anche di 2 o 3 volte; ma il più importante risultato è quello di avere un più affidabile test di zero per la routine di stop del processo iterativo.

Velocità di convergenza

I metodi iterativi, qualunque essi siano, producono un successione di valori $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ – che indicheremo nel seguito con $\{x_n\}$ per $n \in \mathbb{N}$ – convergente alla radice cercata. Ossia tale che:

$$\lim_{n \rightarrow \infty} x_n = \tilde{x} \quad \text{con} \quad f(\tilde{x}) = 0$$

Chiaramente una misura della velocità di convergenza del metodo è data dalla successioni degli errori:

$$e_n = x_n - \tilde{x} \Rightarrow \{e_n\} \text{ per } n \in \mathbb{N}$$

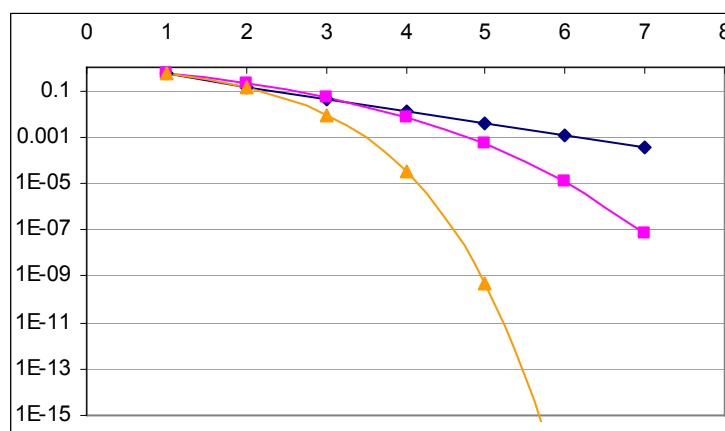
Ad esempio: tre processi iterativi hanno prodotto le seguenti successioni degli errori

n	S ₁	S ₂	S ₃
1	0.5	0.5	0.5
2	0.15	0.1894646	0.125
3	0.045	0.0486978	0.0078125
4	0.0135	0.0072691	3.0518E-05
5	0.00405	0.000507	4.6566E-10
6	0.001215	1.219E-05	1.0842E-19
7	0.0003645	6.598E-08	5.8775E-39

Tutti i processi sono partiti dallo stesso punto iniziale x_0 e di conseguenza hanno lo stesso errore iniziale $e_0 = |x_0 - \tilde{x}|$

Si nota che l'errore del il terzo processo converge più velocemente a zero mentre il primo è il più lento. Intuitivamente la velocità di convergenza non è la stessa.

Inoltre si osserva che il primo processo è più veloce del secondo nelle prime iterazioni, ma viene superato nelle successive. Il grafico seguente mostra chiaramente l'evoluzione degli errori



Ordine e costante di convergenza

Da quanto visto, per confrontare differenti metodi iterativi, storicamente è stato introdotto il concetto di *ordine di convergenza*. Data un successione convergente $\{e_n\}$ per $n \in \mathbb{N}$, diremo che la successione ha ordine di convergenza "p" se esiste ed è finito il seguente limite

$$\lim_{n \rightarrow \infty} \left| \frac{e_{n+1}}{e_n^p} \right| = c \quad (2.3)$$

Il limite "c", se esiste, è definito *costante asintotica* di convergenza.

In particolare si dice che la successione è lineare se $p = 1$, superlineare se $p > 1$. In particolare, si dice quadratica se $p = 2$, cubica se $p = 3$, ecc.

Agli effetti pratici del calcolo, l'ordine di convergenza è legato alle cifre esatte "catturate" ad ogni iterazione secondo lo schema seguente

ordine di convergenza	cifre esatte per iterazione
lineare	costante
superlineare	in progressione
quadratica	raddoppiate

La definizione (2.3) dell'ordine di convergenza implica che per n sufficientemente grande

$$|e_{n+1}| \leq c \cdot |e_n|^p \Rightarrow |x_{n+1} - \tilde{x}| \leq c |x_n - \tilde{x}|^p \quad (2.4)$$

Questa relazione indica che la riduzione dell'errore ad ogni passo è tanto maggiore quanto più alto è l'ordine di convergenza e, a parità di ordine, quanto più piccola è la costante asintotica. L'uso diretto della (2.4) per la deduzione a priori dell'ordine di convergenza è piuttosto laborioso e può essere calcolato solo in casi particolari.

Un esempio è dato dallo schema iterativo per l'estrazione di radice $x = \sqrt[n]{a}$

Partendo dal punto iniziale $x_0 = a$, il processo iterativo converge alla radice n-esima di a

$$x_{i+1} = \frac{1}{n} \left[(n-1)x_i + \frac{a}{x_i^{n-1}} \right], \quad i = 0, 1, 2, \dots$$

Vedremo che questo schema può essere derivato con il metodo generale di Newton. Qui diciamo solo che si tratta di uno schema molto efficiente che viene correntemente usato sui calcolatori per l'estrazione della radice usando solo le comuni operazioni aritmetiche.

Nel caso di radice quadrata (n = 2) la formula iterativa diventa

$$x_{i+1} = \frac{1}{2} \left(x_i + \frac{a}{x_i} \right)$$

Ricordando che $\tilde{x} = \sqrt{a}$, l'errore al passo i+1 è dato da:

$$x_{i+1} - \tilde{x} = \frac{1}{2} \left(x_i + \frac{a}{x_i} \right) - \sqrt{a} = \frac{x_i^2 - 2\sqrt{a}x_i + a}{2x_i} = \frac{(x_i - \sqrt{a})^2}{2x_i}$$

Per $i \gg 1$ $x_i \cong \sqrt{a}$, quindi per la (2.4) si ha

$$|x_{i+1} - \tilde{x}| \leq \left| \frac{(x_i - \sqrt{a})^2}{2x_i} \right| \cong \left| \frac{(x_i - \sqrt{a})^2}{2\sqrt{a}} \right| = \frac{1}{2\sqrt{a}} |x_i - \tilde{x}|^2$$

Ovvero

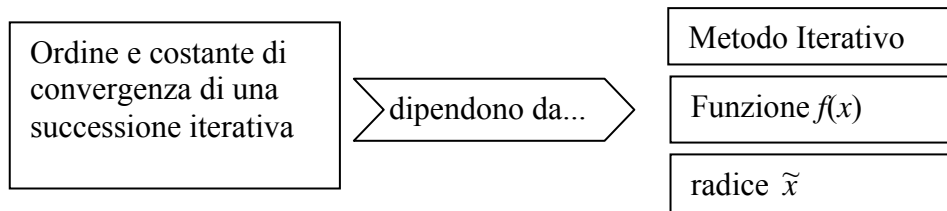
$$e_{n+1} \leq \frac{1}{2\sqrt{a}} \cdot e_n^2$$

Dall'ultima relazione si deduce finalmente che l'ordine di convergenza per l'estrazione di radice quadrata con lo schema iterativo dato è 2, con costante asintotica pari a $1/(2\sqrt{a}) \cong 0.35$

Come si vede il calcolo teorico dell'ordine di convergenza è piuttosto laborioso anche nei casi di schemi iterativi relativamente semplici. In altri casi, come nei casi di radice cubica e quarta non si riesce a risolvere la disequazione (2.3). Oltre a ciò, si deve dire che i metodi iterativi pratici, che vedremo in seguito sono in generale molto più complessi di quello appena descritti. In genere il flusso del processo viene deviato dagli operatori logici decisionali (IF) che attivano differenti

formule iterative a seconda del caso. In queste situazioni il calcolo a priori dell'ordine del metodo è impossibile.

Oltre a ciò premettiamo che anche se ciò fosse possibile, l'ordine di convergenza definito dalla (2.3) varia con il comportamento locale della funzione $f(x)$; cioè in parole povere dipende dalla funzione $f(x)$ e dalla sua radice. Sinteticamente possiamo riassumere dicendo che



Osserviamo che nei casi pratici né la funzione né ovviamente la radice sono conosciute. Può accadere che, come vedremo, lo stesso metodo applicato ad una stessa funzione possa essere molto rapido o molto lento a seconda della radice che stiamo cercando.

Il termine "ordine di convergenza" richiama alla mente qualcosa di fisso, insito nel metodo stesso, come ad esempio le analoghe definizioni di ordine di equazione differenziale. In realtà, da quanto detto, il termine sembra improprio perché in realtà l'ordine di convergenza dipende dal metodo, dalla funzione e dal punto stesso della radice.

Stima dell' ordine di convergenza

Attraverso l'esame della successione degli errori $\{e_n\}$ è possibile per mezzo della (2.4) ottenere un stima dell'ordine e della costante asintotica di convergenza. Diciamo subito che queste stime vanno prese con le dovute cautele per una serie di drastiche assunzioni che nei casi reali devono essere opportunamente verificate. La prima di queste assunzioni è la seguente. Per $i \gg 1$ la relazione (2.4) diventa:

$$|e_{i+1}| \cong c \cdot |e_i|^p \quad , \quad |e_{i+2}| \cong c \cdot |e_{i+1}|^p \quad \dots \quad (2.5)$$

Dividendo si ha

$$\left| \frac{e_{i+1}}{e_{i+2}} \right| = \left| \frac{c \cdot e_i^p}{c \cdot e_{i+1}^p} \right| \Rightarrow \left| \frac{e_{i+1}}{e_{i+2}} \right| = \left| \frac{e_i}{e_{i+1}} \right|^p$$

Da cui passando ai logaritmi

$$p \cong \log \left(\left| \frac{e_{i+1}}{e_{i+2}} \right| \right) / \log \left(\left| \frac{e_i}{e_{i+1}} \right| \right) \quad c \cong |e_{i+1} / e_i|^p \quad (2.6)$$

Queste formule permettono di stimare l'ordine e la costante di convergenza per mezzo di tre valori della successione degli errori $\{x_i - \tilde{x}\}$ dove al posto della vera radice \tilde{x} si sostituisce la miglior approssimazione della successione, in genere l'ultimo valore x_n

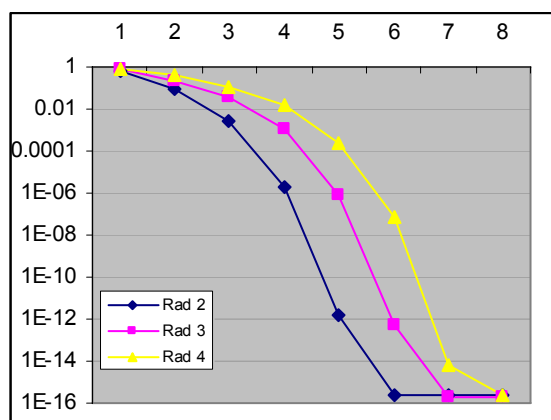
Affinché le formule (2.6) siano attendibili devono essere scelti i valori della successione per $i \gg 1$, o almeno gli ultimi valori della successione. Occorre però fare attenzione ad una cosa. Nei calcoli pratici gli ultimi valori della successione degli errori risentono del limite di precisione di macchina per cui assumono una traiettoria casuale, molto diversa da quella ipotizzata dalle (2.5). Per cui dovremo eliminare dalla successione gli ultimi valori che hanno palesemente raggiunto il limite inferiore (vedi *breakdown*).

Vediamo ora come applicare quanto detto al processo iterativo per l'estrazione di radici. Calcoliamo l'approssimazione delle radici seconda, terza e quarta del numero 2 e applichiamo le formule (2.6) per la stima dell'ordine e la costante di convergenza. Per questo ci serviamo delle funzioni di calcolo di uno spreadsheet

iter.	$\sqrt{2}$		$\sqrt[3]{2}$		$\sqrt[4]{2}$	
	x	err 2	x	err 3	x	err 4
0	2	0.5857864	2	0.740079	2	0.810793
1	1.5	0.0857864	1.5	0.240079	1.5625	0.373293
2	1.41666666666667	0.0024531	1.29629629629630	0.0363752	1.30294700000000	0.11374
3	1.41421568627451	2.124E-06	1.26093222474175	0.0010112	1.20325256868678	0.014045
4	1.41421356237469	1.595E-12	1.25992186056593	8.107E-07	1.18945113368314	0.000244
5	1.41421356237309	2.22E-16	1.25992104989539	5.216E-13	1.18920719008396	7.51E-08
6	1.41421356237309	2.22E-16	1.25992104989487	2E-16	1.18920711500273	7.11E-15
7	1.41421356237309	2.22E-16	1.25992104989487	2E-16	1.18920711500272	2.22E-16

Le curve degli errori sono mostrate nel grafico logaritmico a sinistra. le formule iterative, se pur velocissime, mostrano una netta differenza di velocità di convergenza. Usiamo le formule per stimare l'ordine e la costante di convergenza per tutti e tre i metodi.

Come si vede chiaramente le ultime tre iterazioni di "Rad 2" hanno raggiunto il limite minimo e devono essere scartate. Per lo stesso motivo, le ultime due iterazioni della curva "Rad 3" e l'ultima iterazione di "Rad 4"



Calcoliamo l'ordine di convergenza di $\sqrt{2}$ prendendo gli ultimi errori validi $\{e_4, e_3, e_2\}$

$$p \cong \log(2.124E-6 / 1.595E-12) / \log(0.002453 / 2.124E-6) \cong 2.000$$

$$c \cong 1.595E-12 / (2.124E-6)^2 \cong 0.35$$

Che sono praticamente gli stessi valori già ottenuti per via teorica.

Se ripetiamo il calcolo prendendo altri tre punti consecutivi della successione otteniamo dei valori sensibilmente differenti man mano che ci allontaniamo dalla fine della successione.

Nelle tabelle seguenti sono riportati i valori dell'ordine e la costante di convergenza per tutti e tre i processi e per tutte le possibili terne delle tre successioni (solo i valori validi).

iter	$\sqrt{2}$		$\sqrt[3]{2}$		$\sqrt[4]{2}$	
	p	c	p	c	p	c
2	1.8503	0.2308	1.6762	0.3976	1.5322	0.5148
3	1.9839	0.3204	1.8986	0.5461	1.7600	0.6443
4	1.9997	0.3524	1.9897	0.7386	1.9376	0.9481
5	-	-	1.9999	0.7921	1.9953	1.2122
6	-	-	-	-	2.0040	1.3033

Sorprendentemente tutti i metodi rivelano un ordine quadratico di convergenza ($p = 2$), quello che cambia è la costante asintotica: $c \cong 0.35$ per la radice quadrata, $c \cong 0.79$ per la radice cubica e infine $c \cong 1.3$ per la radice quarta. Si nota anche che i valori ottenuti con i primi valori della successione sono sensibilmente diversi da quelli finali, soprattutto per quanto riguarda la costante asintotica

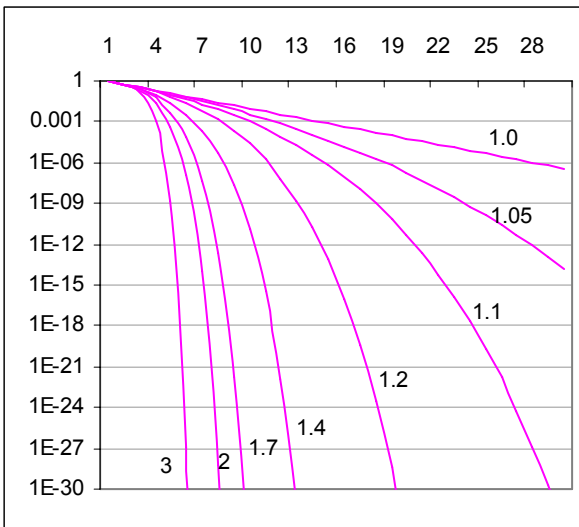
Traiettorie di convergenza

Per mezzo della formula di ricorrenza possiamo tracciare la traiettoria dell'errore di un metodo teorico con differenti velocità di convergenza. partendo da un valore dell'errore unitario $e_0 = 1$

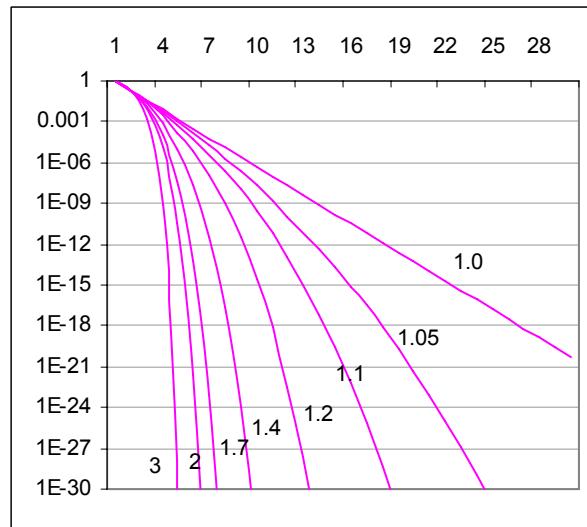
$$|e_{i+1}| = c \cdot |e_i|^p \quad i = 0, 1, 2, \dots \quad (2.7)$$

Le curve dei grafici logaritmici seguenti sono state calcolate per differenti ordini di convergenza (1, 1.05, 1.1, 1.2, 1.4, 1.7, 2, 3) e con due valori della costante asintotica (0.6, 0.2)

costante asintotica $c = 0.6$, $e_0 = 1$



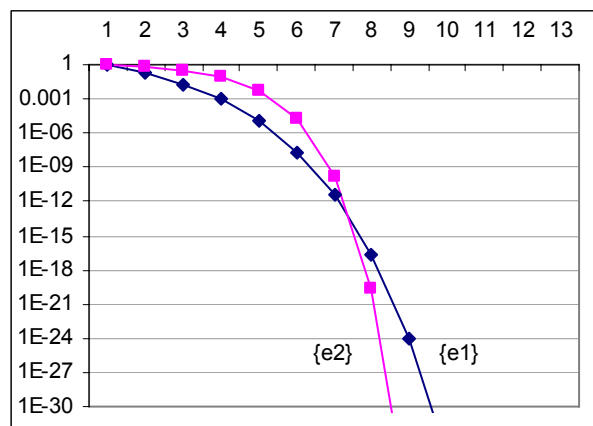
costante asintotica $c = 0.2$, $e_0 = 1$



Come si vede, l'influenza della costante asintotica si riflette soprattutto sulle curve di basso ordine, mentre quelle a più alto ordine, tipicamente $p > 1.7$, rimangono quasi inalterate.

Ci domandiamo se un metodo di alto ordine è sempre più veloce di un metodo di più basso ordine. E' facile mostrare attraverso i grafici che ciò non è sempre vero. Infatti supponiamo che il metodo 1 abbia ordine $p_1 = 1.4$ con costante $c_1 = 0.2$ e che il metodo 2 abbia ordine $p_2 = 2$ con costante $c_2 = 0.7$

c =	0.2	0.7
p =	1.4	2
e0 =	1	1
i	{e1}	{e2}
1	1	1
2	0.2	0.7
3	0.02101	0.343
4	0.0009	0.08235
5	1.1E-05	0.00475
6	2.2E-08	1.6E-05
7	3.9E-12	1.7E-10
8	2.1E-17	2.1E-20
9	9.1E-25	3.2E-40



La successione degli errori innescata dalla formula ricorrente (2.7) mostra che l'errore del primo metodo è sempre minore di quello del metodo di più alto grado fino a valori dell'ordine di $1E-15$.

Poiché, in genere, questo è il limite di macchina di un PC, ne consegue che fino a queste precisioni il metodo 1 è preferibile al metodo 2

Stima statistica dell'ordine di convergenza

Nei casi pratici i metodi iterativi di ricerca degli zeri, seguono abbastanza approssimativamente la formula di ricorrenza dell'errore descritta nei precedenti paragrafi. I motivi sono molteplici quali il comportamento della funzione nell'intorno della radice, la barriera della precisione di macchina, gli errori di arrotondamento, e infine il comportamento decisionale dell'algorithmo stesso che opera in base ai test sui valori correnti e pregressi della successione stessa.

Tipico caso sono come vedremo gli algoritmi cosiddetti "ibridi", quelli costruiti con differenti metodi iterativi aventi caratteristiche complementari. L'algorithmo, passo dopo passo, sceglie in base a certi test quale è conveniente usare.

Il risultato macroscopico è un certa componente aleatoria, più o meno evidente, che si sovrappone alla successione degli errori $\{e_n\}$, per cui le formule ricavate nel precedente paragrafo possono fallire vistosamente, specialmente per quanto riguarda la costante asintotica.

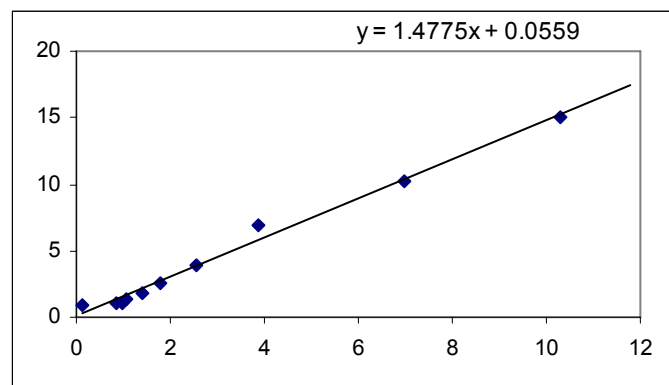
Per ovviare a questo si può ricorrere al metodo statistico. Riprendiamo la formula di ricorrenza dell'errore (2.7) e passando ai logaritmi si ottiene

$$|e_{i+1}| = c \cdot |e_i|^p \Rightarrow \log(|e_{i+1}|) = \log(c \cdot |e_i|^p) \Rightarrow \log(|e_{i+1}|) = \log(c) + p \cdot \log(|e_i|)$$

quindi passando in coordinate logaritmiche $t_i = -\log(|e_i|)$ e $u_i = -\log(|e_{i+1}|)$ si ha $u_i = q + p \cdot t_i$ che è una retta nel RCO (t, u). I parametri "q" e "p" possono essere trovati con la regressione lineare con il criterio dei minimi quadrati.

Vediamo come funziona questo metodo per mezzo di un esempio pratico. La successione seguente è tratta da un algoritmo reale¹

$ e_i $	t_i	u_i
0.7169852	0.14	0.87
0.1352012	0.87	0.99
0.1014803	0.99	1.08
0.0823848	1.08	1.39
0.0406458	1.39	1.78
0.0165823	1.78	2.57
0.0026873	2.57	3.85
0.0001407	3.85	6.97
1.074E-07	6.97	10.29
5.13E-11	10.3	15
1E-15		



Anche qui potremmo procedere allo scarto dei punti palesemente fuori statistica. In questo esempio non ce ne sono quindi accettiamo tutti i punti ed eseguiamo la regressione lineare su 10 punti. il risultato è:

$$u = 1.477 t + 0.0559.$$

Da cui

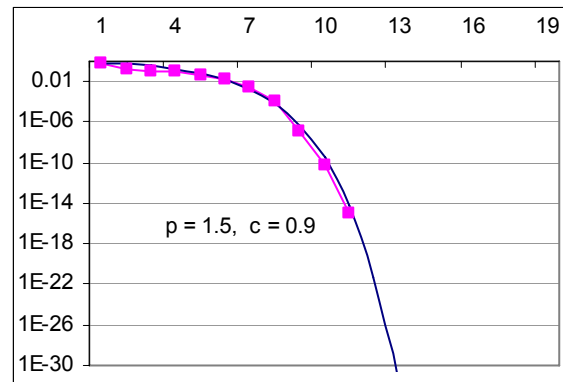
$$p \cong 1.5 \quad , \quad c \cong 10^{-0.0559} \cong 0.88$$

¹ si tratta del cosiddetto algoritmo Wijngaarden-Dekker-Brent, uno dei più efficaci e robusti rootfinder. Si tratta di un algoritmo ibrido perché usa una combinazione di tre metodi differenti: bisezione, "regula falsi" e interpolazione quadratica inversa.

Quindi l'ordine di convergenza medio del metodo è di circa 1.5 con una costante asintotica di circa 0.88. Si deve notare che l'applicazione diretta delle formule (2.6) a questa sequenza avrebbe dato dei risultati incoerenti.

Vediamo invece se i risultati trovati sono accettabili. Per questo dobbiamo innescare la formula iterativa degli errori (2.7) con i parametri $e_0 = 0.72$, $p = 1.5$, $c = 0.88$ ed osservare se e in che misura il grafico si sovrappone ai punti della successione data $\{e_0, e_1, \dots, e_{11}\}$

I punti della successione (rosa) si sovrappongono alla curva degli errori di parametri (0,72, 1.5, 0.88) con buona precisione generale. Pertanto possiamo accettare i valori trovati sia per l'ordine che per la costante asintotica di convergenza del metodo. Si deve ricordare che ciò vale per la specifica funzione e la radice trovata. Per un'altra coppia di funzione-radice il metodo può benissimo avere altri valori.



Osservazione. dalle prove sperimentali si è visto che il parametro "c", la costante asintotica di convergenza, è quello che risente in misura maggiore dalle perturbazioni per cui la sua precisione è piuttosto scarsa (10 - 25 %), mentre, al contrario, l'ordine di convergenza "p" è molto più preciso (3-8%). Per questo motivo, a volte, è necessario aggiustare la costante asintotica c al fine di migliorare la sovrapposizione delle due curve. Pochi tentativi sono necessari in genere per trovare il valore per un giusto compromesso

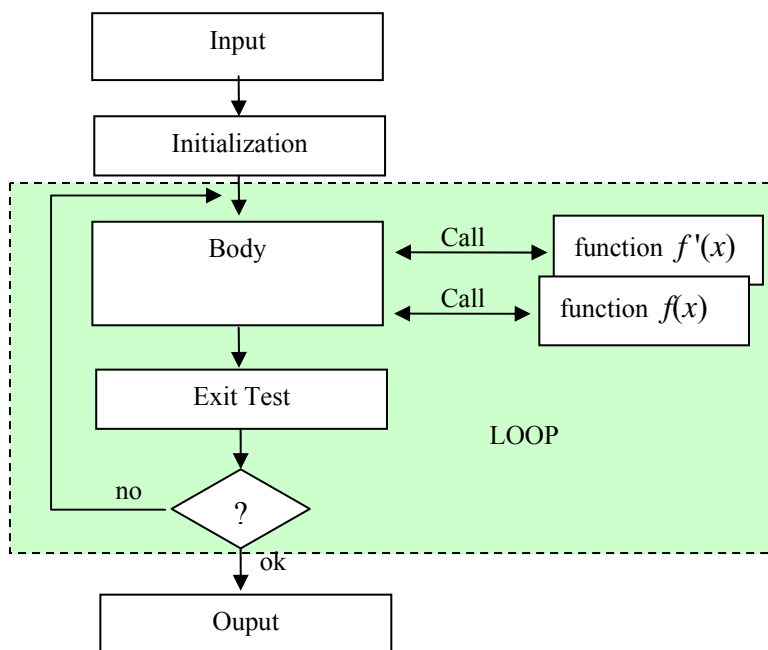
Algoritmi

Tutti gli algoritmi iterativi vengono realizzati, praticamente, mediante programmi automatici chiamati procedure o routine. Il calcolo della funzione e delle sue eventuali derivate viene invece effettuato da altri programmi esterni (function) e richiamati dal programma principale quando è necessario. La routine "rootfinder" che esegue il metodo iterativo conosce quindi la funzione $f(x_i)$ e le sue eventuali derivate $f'(x_i)$, $f''(x_i)$, ecc. solamente attraverso i valori ritornati dalle routine function. Ad eccezione di casi particolari come i polinomi, la routine rootfinder non conosce la struttura matematica della funzione di cui vuole approssimare gli zeri.

Flow-chart

In generale gli algoritmi vengono rappresentati con diagrammi di flusso (flow-chart). Per gli algoritmi iterativi che stiamo studiando, il relativo diagramma è molto semplice e si compone di 4 parti principali:

- Lettura parametri ingresso (input).
- Dimensionamento e scelta dei valori iniziali (Initialization)
- Ciclo iterativo (loop)
- Scrittura risultati (output)



La parte del loop è il cuore dell'algoritmo iterativo.

- All'interno di esso vengono eseguite e calcolate le formule d' iterazione.
- Vengono richiamate le "function" per il calcolo della funzione ed eventuali derivate;
- Si trovano i controlli per il test di fine-loop.
- Vengono ri-inizializzate le variabili con i valori correnti prima dell'iterazione successiva

La parte interna del "loop" viene ripetuta più volte, quindi è quella che costa di più in termini di tempo di elaborazione. Lo sviluppatore deve valutare molto bene il tempo di esecuzione per ciclo mantenendolo più basso possibile perché da esso dipendono in massima parte le qualità dell'intero programma. I modi per farlo sono due: diminuendo il numero d'iterazione e diminuendo il tempo per ciascun iterazione. L'adozione di algoritmi con alto ordine di convergenza ma con numero di calcoli per ogni iterazione molto pesante potrebbero essere altrettanto sfavorevoli quanto un algoritmo a lenta convergenza

Da quanto detto emerge la necessità per chi sviluppa di tenere in considerazione anche il costo dell'algoritmo e non solo la sua velocità di convergenza. Regole precise e standard non sono ancora state definite in letteratura. D'altra parte una serie di norme per il confronto degli algoritmi da

questo punto di vista è necessaria. Nei paragrafi seguenti useremo quindi una metrica, molto semplice, per la valutazione del costo computazionale degli algoritmi

Costo computazionale

In questo paragrafo vogliamo definire una metrica per il costo computazione allo scopo di confrontare differenti algoritmi.

Come linea guida ci serviremo del tempo di elaborazione dei moderni PC, stabilendo che il *costo computazione* è inversamente proporzionale al tempo di elaborazione per ogni iterazione.

Al fine di semplificare la metrica del costo faremo anche delle opportune (drastiche) semplificazioni dettate dall'osservazione pratica

Un tempo le operazioni aritmetiche elementari erano effettuate dagli elaboratori in tempi diversi: La somma e la sottrazione erano le più veloci seguite dalle moltiplicazione e poi dalla divisione e dall'estrazione di radice. Era prassi comune quindi contare il numero delle moltiplicazione e divisioni di una routine per stimare il suo costo computazionale. Si era soliti dire: "è un algoritmo da 10 moltiplicazioni e 3 divisioni". Sui moderni calcolatori, grazie a particolari circuitazioni elettroniche e firmware/software speciali (un esempio è l'algoritmo iterativo di estrazione di radice già visto) queste operazioni sono diventati ormai comparabili.

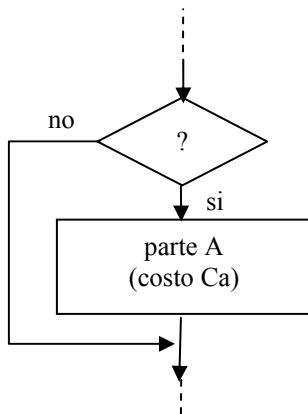
Su questa ipotesi possiamo fissare l'unità di misura del costo computazionale quella di un operazione aritmetica elementare, con simbolo "op". Non ci preoccupiamo di stabilire quantitativamente questo valore perché useremo questa unità solo per confronto. Diremo cioè "l'algoritmo A costa 5 operazioni (5 op), l'algoritmo B costa 15 op" intendendo con questo che l'algoritmo B effettua il triplo delle operazione aritmetiche di quello di A per ogni iterazione. Sotto queste assunzioni il costo dell'algoritmo può essere addirittura conteggiato a partire dal suo codice o, con opportune semplificazioni, dalla sua formula iterativa.

Regole di costo computazionale

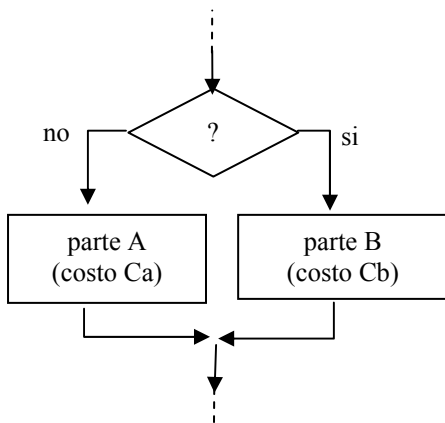
Conteggiare solo le espressioni aritmetiche elementari contenute nel ciclo.	
Conteggiare 1 "op" le operazioni aritmetiche elementari (somma, sottrazione, moltiplicazione, divisione, potenza, radice)	+ - * / \sqrt{x} $\sqrt[n]{x}$ x^n
Non conteggiare le funzioni di segno (valore assoluto, segno, opposto)	$ x $, segno(x), -x,
Non conteggiare le assegnazioni di variabili	$x = 3.5$, $x = y$
Non conteggiare le condizioni logiche (if) se la condizione non contiene operazioni	$\text{if}(x > 0)$, $\text{if}(x = 0)$, $\text{if}(x < 0)$
Si contano invece le operazioni contenute implicitamente nelle condizioni logiche	$\text{if}(x < 3) \rightarrow \text{if}(x-3 < 0) \rightarrow 1 \text{ op}$ $\text{if}(e*x > \text{tol}) \rightarrow \text{if}(e*x-\text{tol}>0) \rightarrow 2 \text{ op}$
Contare le operazioni contenute nei rami logici moltiplicate per la rispettiva probabilità di evento	Nel caso più comune che la probabilità non possa essere determinata a priori si usa il principio di probabilità uniforme (1/2)

L'ultima regola, se pur d'uso meno frequente, necessita di qualche chiarimento

Possibili flussi logici condizionati



Il contributo effettivo della parte A viene ridotto di un fattore pari a due, perché la probabilità che il flusso possa seguire il ramo "si" è pari a 0.5 .
 Ad esempio se la parte A contiene la formula
 $y = (2*a+5*x)/(x+1)$
 avente costo $C_a = 5$ op
 il contributo effettivo al costo totale è pari a $5/2$



Il contributo effettivo della parte A e B vengono ridotti di un fattore pari a due, perché la probabilità che il flusso possa seguire il ramo "si" o il ramo "no" è pari a 0.5
 Ad esempio se la parte A ha costo $C_a = 5$ op
 e se la parte B ha costo $C_b = 11$ op
 il contributo effettivo al costo totale è pari alla media di C_a e C_b . Infatti
 $C = 0.5*C_a + 0.5*C_b = (C_a+C_b)/2 = 16/2 = 8$ op

Si deve notare che un eventuale espressione nella condizione logica (rombo) deve invece essere conteggiata per intera perché essa viene sempre valutata a prescindere dal risultato della condizione.

Costo del calcolo della funzione

E' evidente che anche il calcolo della funzione ha un costo computazionale; anzi in alcuni casi il tempo di calcolo di $f(x)$ può essere addirittura prevalente rispetto al resto

D'altra parte, dal punto di vista dell'algorithm, la struttura algebrica della funzione non è nota o difficilmente valutabile (si pensi ad una espansione in serie, od una formula integrale)

Per questo si conviene di assumere un seconda unità di misura che tenga conto del costo di calcolo della funzione e scriveremo 1 "f" se un algoritmo richiede il calcolo della $f(x)$ per ogni iterazione; 2 "f" se richiede il calcolo di due volte la $f(x)$ per ogni iterazione, ecc.

Il calcolo delle eventuali derivate viene equiparato al calcolo della funzione stessa.

Quindi completiamo le regole di conteggio con la seguente

Conteggiare 1 "f" ogni calcolo della $f(x)$ o delle sue derivate per iterazione

Quindi riepilogando, diremo che il costo di un algoritmo è pari a $(18 \text{ op} + 1 \text{ f})$ oppure $(10 \text{ op} + 2 \text{ f})$, ecc. Tuttavia per effettuare dei confronti, quando possibile, ci conviene avere un equivalenza fra "f" e "op". Da risultati statistici è emerso che il fattore di conversione "1 f = 5 op" rende abbastanza confrontabili i due costi per funzioni non particolarmente gravose. Invece per funzioni complesse (ottenute da serie o integrazioni numeriche) il fattore 1:10 è più equilibrato.

Calcolo delle Derivate

I metodi iterativi possono dividersi in due famiglie: quelli che usano le derivate della funzione $f(x)$ e quelli che non le usano. Gli algoritmi che usano i valori di $f(x)$, $f'(x)$, ecc. hanno in generale un ordine più elevato degli altri. E quindi si tratta di algoritmi molto veloci. Fanno parte di questa categoria i metodi di Newton-Raphson, Halley, Chebyshev, Householder, e molti altri

Nei moderni calcolatori, tuttavia, l'operazione di derivata non è disponibile se non attraverso l'acquisizione di speciali programmi software¹. E' per questo motivo che recentemente hanno acquistato importanza anche altri metodi, i cosiddetti "derivatives-free", che risolvono il problema delle radici in modo altrettanto efficiente, senza ricorrere all'uso delle derivate.

Questo aspetto è molto importante dal punto di vista numerico ed è una delle prime cose che devono essere considerate in fase di sviluppo delle routine "rootfinder".

Un'altra possibilità è quella di approssimare i valori delle derivate mediante le formule alle differenze divise centrate.

$$f'(x_i) \cong \frac{f(x_i + h) - f(x_i - h)}{2h}$$

$$f''(x_i) \cong \frac{f(x_i + h) - 2f(x_i) + f(x_i - h)}{h^2}$$

Queste formule sono sufficientemente precise per la maggior parte dei problemi che si incontrano nella pratica. Il parametro "h" non è critico e, se la radice è semplice, può essere scelto entro ampi limiti; valori comunemente usati vanno da 1E-4 a 1E-7.

Le formule di approssimazione sono utili anche quando il calcolo della derivata simbolica risulterebbe troppo laborioso (es. $f(x) = \Gamma(x+1)/\text{erf}(x)$).

Bisogna sottolineare che approssimare le derivate non significa rinunciare alla precisione globale della radice. Come vedremo in seguito, nel caso di radici semplici, la precisione nel calcolo delle derivate non influenza la precisione del valore finale che dipende solamente dal calcolo di $f(x)$. Per le radici multiple l'influenza delle derivate inizia a farsi sentire e bisognerebbe ridurre il valore di h progressivamente con le iterazioni.

L'unico vero difetto delle formule di approssimazione è il loro costo. Infatti l'approssimazione della derivata prima richiede il calcolo di 2 volte $f(x)$, e la derivata seconda di 2 volte $f(x)$ (il valore $f(x_i)$ è già disponibile dal calcolo della funzione stessa).

Quindi ricapitolando, gli aspetti importanti da tenere in considerazione sono

Metodi con derivate	Metodi senza derivate
<ul style="list-style-type: none"> • Alta velocità di convergenza • Nel calcolo automatico necessitano di speciali programmi (CAS) • Moderata complessità di $f(x)$ • Alto costo computazionale per l'approssimazione delle derivate 	<ul style="list-style-type: none"> • Moderata velocità di convergenza • Non richiedono speciali programmi • Nessun limite per la complessità di $f(x)$ • Basso costo computazionale

¹ Programmi software per la derivata simbolica, chiamati CAS

Metodo del Punto Fisso

Si tratta di un metodo molto popolare, ancora usato nella didattica per la dimostrazione di numerose proprietà dei metodi d'iterazione funzionale. Nel campo numerico, viene usato talvolta solo nel calcolo manuale specialmente unito al metodo di accelerazione di Aitken.

Il metodo trova la radice r di un'equazione del tipo $x = G(x)$ mediante l'iterazione

$$x_{i+1} = G(x_i)$$

Condizione sufficiente perché la successione $x_0, x_1, x_2, \dots, x_n$ converga alla radice è:

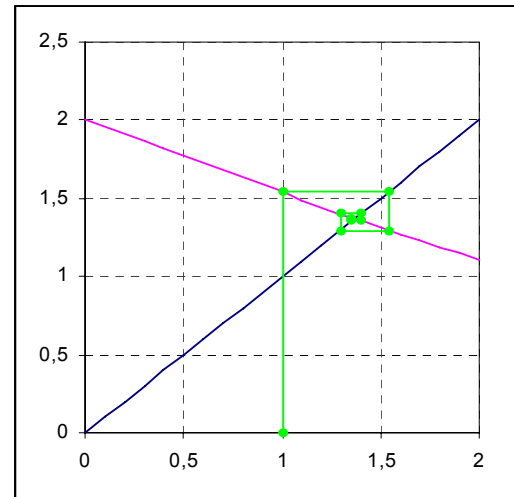
$$|G'(x)| \leq L < 1$$

Cioè la $g(x)$ è una contrazione in un intorno locale della radice. Si definisce l'errore:

$$e_{n+1} = |r - x_n|$$

Allora si dimostra che:

$$e_{n+1} \approx |G'(r)| e_n$$



Da questa ultima relazione si vede che l'errore si riduce ad ogni iterazione di un fattore prossimo ad $|G'(r)|$. Se $|G'(r)|$ è vicino ad 1 si avrà una lenta riduzione dell'errore e quindi anche una lenta convergenza. Vediamo come funziona questo metodo con un esempio.

Esempio: Nell'anno 1225 il matematico Leonardo Pisano Fibonacci studiò la seguente equazione cubica:

$$x^3 + 2x^2 + 10x - 20 = 0$$

e calcolò l'unica radice reale $x = 1.368808107$.

Nessuno conosce quale metodo Fibonacci usò per trovare questo valore, ma è un risultato notevole per il suo tempo. Per mezzo della formula di Cardano-Tartaglia è possibile esprimerla in modo esatto come

$$\tilde{x} = \sqrt[3]{\frac{2\sqrt{3930} + 252}{3} + \frac{252}{27}} - \sqrt[3]{\frac{2\sqrt{3930} - 252}{3} - \frac{2}{27}} - \frac{2}{3}$$

Approssimando questa formula con 15 cifre significative otteniamo.

$$x \cong 1.36880810782137$$

Come possiamo verificare il risultato calcolato da Fibonacci ha ben 10 cifre esatte! E', senza dubbio, un eccellente risultato anche oggi.

Usiamo il metodo del punto fisso per trovare la radice. Prima portiamo l'equazione nella forma adatta. Vi sono vari modi, ma, come vedremo, non tutti sono validi. Poniamo, ad esempio:

$$x(x^2 + 2x + 10) = 20$$

Per i valori di x tale che $x^2 + 2x + 10 \neq 0$ possiamo scrivere:

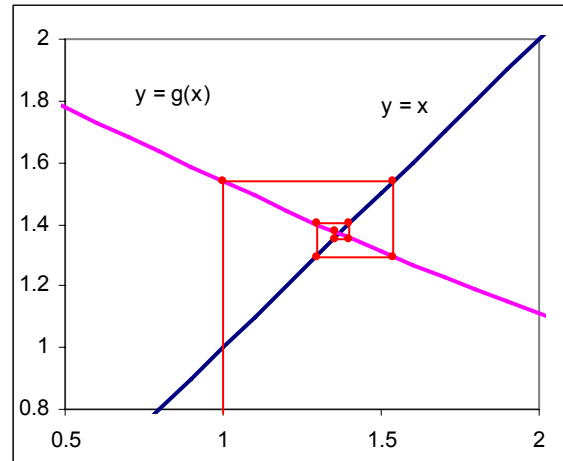
$$x = \frac{20}{(x^2 + 2x + 10)}$$

Ponendo

$$G(x) = \frac{20}{(x^2 + 2x + 10)}$$

otteniamo la funzione iterativa $G(x)$. Scegliamo come punto d'innescio, ad esempio, $x_0 = 1$

x	g(x)
1	1.538461538
1.538461538	1.295019157
1.295019157	1.401825309
1.401825309	1.35420939
1.35420939	1.375298092
1.375298092	1.365929788
1.365929788	1.370086003
1.370086003	1.368241024
1.368241024	1.369059812
1.369059812	1.368696398
1.368696398	1.368857689
1.368857689	1.368786103
1.368786103	1.368817874



Come si nota, per raggiungere la precisione di 4 cifre si deve arrivare alla 13 iterazione. Stimiamo la velocità di convergenza. Calcoliamo, innanzitutto, la derivata di $G(x)$

$$G'(x) = \frac{40(x+1)}{(x^2 + 2x + 10)^2}$$

Da cui, sostituendo un valore approssimato della radice, abbiamo $G'(1.3688) \cong -0.4447...$

Quindi: $e_{n+1} \approx 0.44 e_n$. Questo significa che saranno necessarie circa 3 iterazioni per ottenere una cifra esatta. Per 4 cifre saranno necessarie circa 12 iterazioni; che è proprio quanto l'algoritmo ha prodotto.

Faremo vedere ora che altri arrangiamenti dell'equazione possono non essere altrettanto validi. Il più immediato è il seguente:

$$x = \frac{1}{10}(-x^3 - 2x^2 + 20) \Rightarrow G(x) = \frac{1}{10}(-x^3 - 2x^2 + 20)$$

La derivata di $G(x)$ è:

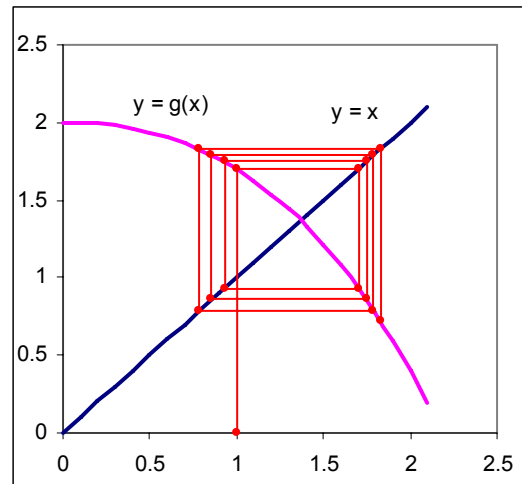
$$G'(x) = -\frac{x(3x+4)}{10}$$

Da cui, sostituendo si ha $G'(1.3688) \cong -1.098...$

Come si vede, in prossimità della radice la derivata è maggiore di 1. Quindi non ci sono speranze di ottenere una successione convergente.

E' interessante, in questo caso, vedere la traiettoria dei punti della successione

x	g(x)
1	1.7
1.7	0.9307
0.9307	1.746142
1.746142	0.8577968
0.8577968	1.7897189
1.7897189	0.7861175
0.7861175	1.8278233
1.8278233	0.7211479
0.7211479	1.8584855
1.8584855	0.6672913
0.6672913	1.8812315
1.8812315	0.6264198
0.6264198	1.8969388



E' evidente il caratteristico andamento di una successione oscillatoria e divergente allo stesso tempo. I valori della successione $x_0, x_1, x_2, x_3, \dots$, che nel grafico sono le ascisse dei punti della "ragnatela" si dirigono alternativamente in direzioni opposte. Il carattere della successione può cambiare man mano che i punti si allontanano dal punto fisso, che è l'intersezione delle due curve. Di una cosa però possiamo essere certi: che in un intorno della radice la successione è senz'altro divergente, e quindi non esiste nessun punto d'innesco che possa far avvicinare la successione alla radice.

Come abbiamo visto la scelta della funzione iterativa $G(x)$, pur arbitraria, può determinare il successo o meno del processo. E' forse questo, piuttosto che la lentezza, il principale punto debole del metodo. L'arbitrarietà della scelta della funzione iterativa mal si presta ad un implementazione automatica.

Accelerazione di Aitken

La velocità del metodo del punto fisso non è elevata. Il metodo di accelerazione di Aitken (chiamato anche processo Δ^2) può, in certe circostanze, accelerare il processo iterativo

Dati 3 valori consecutivi della successione: x_n, x_{n+1}, x_{n+2} , si calcolano le seguenti differenze finite di primo e secondo ordine:

$$\begin{aligned} \Delta_{n+1} &= x_{n+2} - x_{n+1} \\ \Delta_n &= x_{n+1} - x_n \\ \Delta_n^2 &= \Delta_{n+1} - \Delta_n = x_{n+2} - 2x_{n+1} + x_n \end{aligned}$$

Si estrapola il valore finale della successione con la formula:

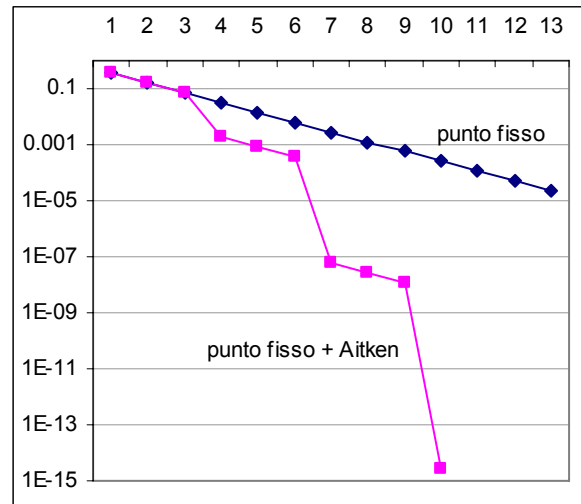
$$x^* = x_{n+2} - \frac{(\Delta_{n+1})^2}{\Delta_n^2} = x_{n+2} - \frac{(x_{n+2} - x_{n+1})^2}{x_{n+2} - 2x_{n+1} + x_n}$$

Se tutto funziona, il valore x^* è un valore approssimato della soluzione vera, che si sarebbe potuto ottenere solo con un numero di iterazioni assai maggiore. Questo valore può essere usato per innescare una nuova successione da cui ottenere 3 nuovi punti. Poi si riapplica la formula di Aitken per ottenere un valore più preciso e così via.

Vediamo come funziona questo metodo sulla successione del punto fisso ottenuta precedentemente

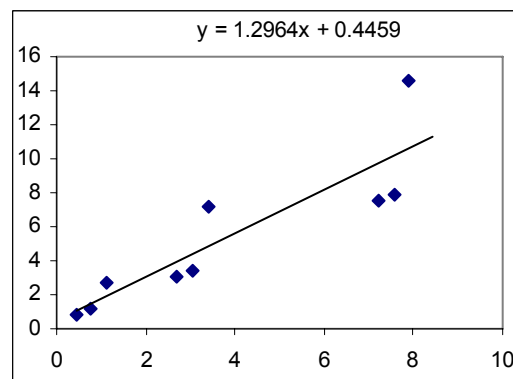
x	G(x)	en
1	1.538461538	0.368808108
1.538461538	1.295019157	0.169653431
1.295019157	1.401825309	0.073788951
1.370813883	1.36791809	0.002005775
1.36791809	1.369203163	0.000890018
1.369203163	1.368632779	0.000395055
1.36880817	1.36880808	6.21234E-08
1.36880808	1.36880812	2.75721E-08
1.36880812	1.368808102	1.22373E-08
1.368808108	1.368808108	2.66454E-15

In giallo sono evidenziati i punti ottenuti con la formula Δ^2 di Aitken
 L'accelerazione è così evidente da non lasciare alcun dubbio sulla bontà del metodo



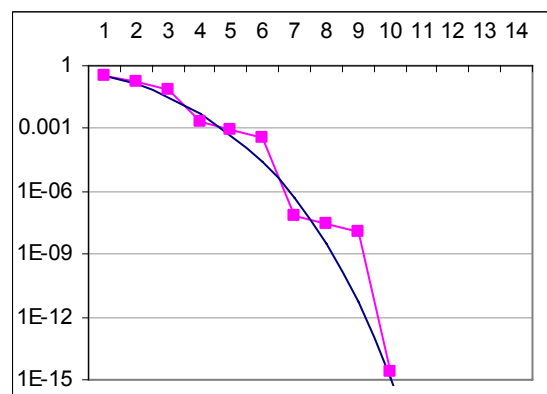
In soli 10 iterazioni il metodo è arrivato alla precisione di circa 1E-15 mentre con solo il punto fisso avremmo ottenuto, a parità di passi, solamente una precisione di appena 1E-3
 E' da notare anche la caratteristica forma "a salti" impressa alla traiettoria dall'accelerazione Δ^2 . Ad ogni salto vengono catturate un numero di cifre sempre maggiore. Questo indica chiaramente che l'ordine di convergenza è senz'altro maggiore di 1.
 Abbiamo già visto che la formula ricorsiva dell'errore per il metodo del punto fisso applicato all'equazione di Fibonacci è : $e_{n+1} \approx 0.44 e_n$. Quindi ne deduciamo che l'ordine di convergenza è pari a 1 con costante asintotica di 0.44; questi valori sono confermati dall'andamento rettilineo della traiettoria.

Per il metodo "punto fisso + Δ^2 " adoperiamo il metodo statistico illustrato nei paragrafi precedenti
 Passando ai logaritmi e costruendo la regressione lineare della tabella $[-\log(e_i), -\log(e_{i+1})]$, con $i = 0,1,2,\dots,9$, otteniamo la retta
 $y \approx 1.3x + 0.44$



Da cui si ottiene l'ordine di convergenza di circa $p \approx 1.3$ e costante $c \approx 10^{-0.44} \approx 0.36$. In realtà, per tentativi, si trova una migliore centratura della traiettoria con il valore $c = 0.45$.

Passando da un ordine di convergenza medio di circa 1 ad un altro con appena 1.3 si ha l'eccellente accelerazione che abbiamo appena visto. Chiaramente guadagnare anche qualche decimo sull'ordine di convergenza dei processi iterativi porta un enorme vantaggio in termini di velocità. Tutti i metodi che vedremo cercano di sfruttare al meglio questo aspetto.



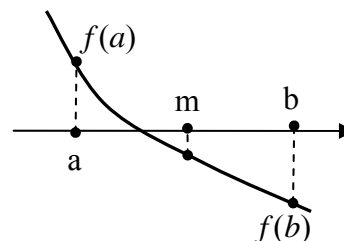
Metodo di Bisezione

Il metodo di bisezione, o dicotomico, è uno dei più semplici metodi iterativi per approssimare gli zeri di una funzione, e in tale semplicità, sta, a nostro avviso, la sua maggior forza. Il metodo è basato sul fatto che se una funzione continua cambia segno in un intervallo, allora in tale intervallo deve esserci almeno uno zero.

Il metodo procede nel seguente modo.

Si sceglie un intervallo $[a, b]$ in cui i segni della funzione agli estremi sono diversi; cioè deve essere $f(a) \cdot f(b) < 0$

L'intervallo viene suddiviso in due parti uguali dal punto medio $m = (a + b)/2$. La radice si deve trovare necessariamente in uno dei due semi-intervalli $[a, m]$ o $[m, b]$. Per saperlo basta calcolare la funzione $f(m)$.



Si possono avere tre casi:

1. $f(m) = 0$ m è la radice cercata, e il processo termina
2. $f(a) \cdot f(m) < 0$ i segni sono discordi e quindi la radice si trova in $[a, m]$
3. $f(a) \cdot f(m) > 0$ i segni sono concordi e quindi la radice si trova in $[m, b]$

Nota bene. I casi 2) e 3) sono complementari per cui basta fare il test su uno per avere informazioni anche sull'altro. Nella figura l'intervallo che contiene la radice è $[a, m]$ che è ridotto di un fattore 2 rispetto all'intervallo di partenza. Ora possiamo ripetere il processo con il nuovo intervallo $[a, m]$ ottenendo un nuovo intervallo, contenente la radice e ridotto di un fattore 2. Ad ogni iterazione l'intervallo contenente la radice si riduce di un fattore 2 secondo la relazione

$$|e_{i+1}| = |x_{i+1} - \tilde{x}| \leq \frac{|b-a|}{2^i}$$

Questa relazione dimostra la convergenza del metodo iterativo essendo

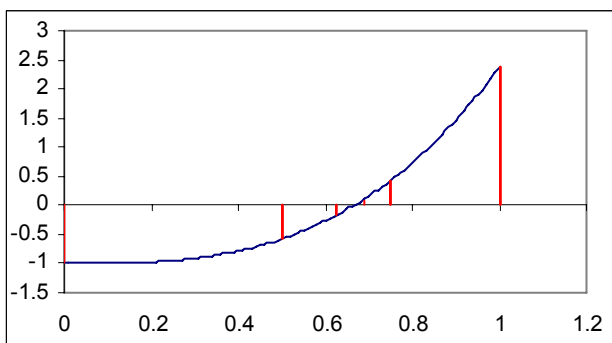
$$\lim_{i \rightarrow \infty} \frac{|b-a|}{2^i} = 0 \Rightarrow \lim_{i \rightarrow \infty} |x_{i+1} - \tilde{x}| = 0 \Rightarrow \lim_{i \rightarrow \infty} x_{i+1} = \tilde{x}$$

per $i \gg 1$ la relazione può scriversi $|e_{i+1}| \approx 0.5|e_i|$ tipica di un processo iterativo di ordine $p = 1$ e costante asintotica di convergenza $c = 0.5$. In pratica viene catturata un cifra significativa ogni 2-3 iterazioni.

Vediamo come il metodo funziona applicato alla funzione

$$f(x) = (3x/2)^3 - 1$$

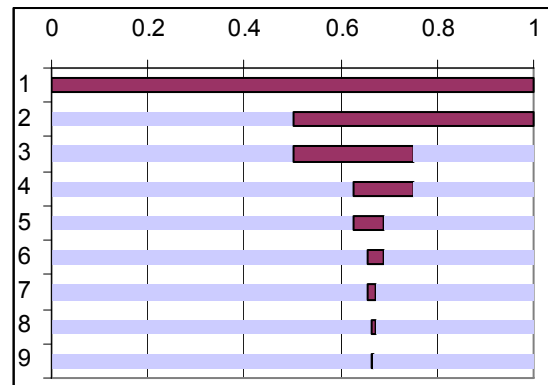
Il cui grafico è riportato a destra
 Assumiamo come intervallo di partenza il segmento $[0, 1]$ in cui si verifica sicuramente la condizione $f(a) \cdot f(b) < 0$ essendo $f(0) = -1$ e $f(1) \cong 2.375$



Il primo punto medio è $m = 0.5$ e $f(0.5) \cong -0.578125$. Il valore della funzione è negativo per cui la radice deve trovarsi fra $0.5 < x < 1$. Il limite $a = 0$ viene quindi sostituito con $a = m = 0.5$, per cui il segmento da cui ripartire per una nuova iterazione è $[0.5, 1]$

Riportiamo qui le prime 9 iterazioni del metodo di bisezione. Nel grafico a sinistra è evidenziata (in scuro) la successione degli intervalli $[a_i, b_i]$ di ogni iterazione.

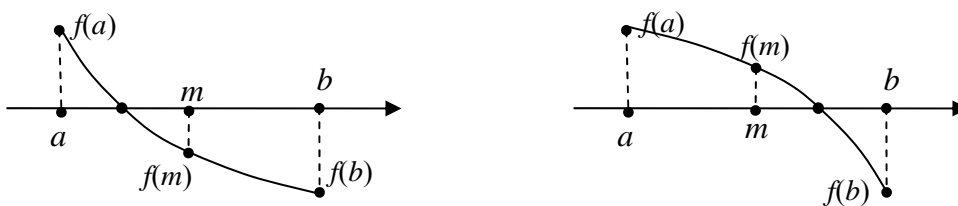
a	f(a)	b	f(b)
0	-1	1	2.375
0.5	-0.578125	1	2.375
0.5	-0.578125	0.75	0.423828
0.625	-0.176025	0.75	0.423828
0.625	-0.176025	0.6875	0.09671
0.65625	-0.046146	0.6875	0.09671
0.65625	-0.046146	0.67188	0.023621
0.664063	-0.011673	0.67188	0.023621
0.664063	-0.011673	0.66797	0.005871



Il metodo di bisezione appartiene alla famiglia dei metodi che "non possono fallire" la cui convergenza alla radice è sempre garantita matematicamente al 100% se sono soddisfatte le condizioni iniziali: continuità della funzione e differenza di segno agli estremi dell'intervallo. Quando l'intervallo contiene più di una radice, il metodo di bisezione converge comunque ad una delle due.

Dettagli implementativi

Per motivi di efficienza, il test sul prodotto $f(a) \cdot f(m) < 0$ viene evitato nelle routine reali. In sostituzione viene effettuato il test del segno del solo punto medio $f(m)$. Perché questo funzioni, all'inizio della routine, fuori dal loop, viene "fissato", il segno dei due estremi: a sinistra il valore positivo e a destra quello negativo. In caso contrario i due estremi vengono scambiati. Poiché l'algoritmo di bisezione non inverte mai gli estremi, per tutto il ciclo d'iterazione, il valore positivo rimarrà sempre a destra e quello negativo a sinistra.



In questa situazione, per decidere dove si trova la radice basta semplicemente controllare il segno di $f(m)$; se negativo la radice è a sinistra del punto m altrimenti sarà a destra. Tutto il cuore dell'algoritmo quindi può essere condensato in poche linee¹.

```

m = (a+b)/2           'calcolo il punto centrale
fm = f(m)            'calcolo la funzione
if fm < 0 then
    b = m, fb = fm
else
    a = m, fa = fm
endif
    
```

Costo computazionale. Il costo del metodo è il più basso di tutti: $2 \text{ op} + 1 \text{ f}$

¹ Il codice completo si trova in appendice

Nessun altro algoritmo può vantare un costo minore. Esiste anche un'altra caratteristica molto importante che rende questo metodo tuttora valido specialmente nel calcolo automatico ed è la sua eccezionale ed intrinseca robustezza. Vediamo un esempio

Sia data la funzione

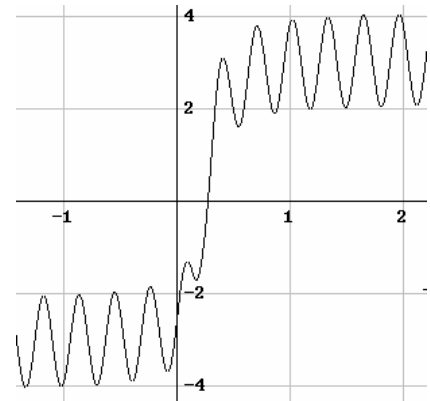
$$f(x) = \operatorname{atan}(10x-3) + \operatorname{atan}(20x-4) + \sin(20x)$$

Cerchiamo lo zero nell'intervallo $[-2, 2]$.

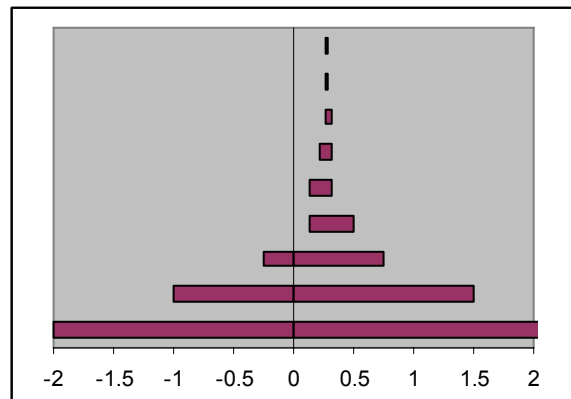
Notiamo come la curva presenti delle oscillazioni molto accentuate sia a sinistra che a destra della radice.

Pochissimi algoritmi saprebbero approssimare la radice partendo da un intervallo così ampio e con tali variazioni.

Il metodo di bisezione, solido come una roccia, invece non ha particolari difficoltà a trovare la radice $x \approx 0.273910\dots$



a	b
-1	2
-1	0.5
-0.25	0.5
0.125	0.5
0.125	0.3125
0.21875	0.3125
0.265625	0.3125
0.265625	0.289063
0.265625	0.277344
0.271484	0.277344



Per queste doti di eccellente robustezza il metodo di bisezione è ancora usato nei moderni rootfinder algoritmi come "apripista" a metodi più veloci ma più instabili al tempo stesso. Tali metodi "ibridi" sono composti in generale da due stadi: Il primo fa partire un metodo lento ma robusto per isolare la radice con una modesta approssimazione mentre il secondo fa partire un metodo ad alto ordine di convergenza per incrementare la precisione fino ai valori richiesti. Il primo stadio è quasi sempre affidato al metodo di bisezione.

Un altro vantaggio del metodo di bisezione è che il numero d'iterazioni è indipendente dalla forma della funzione $f(x)$ ma dipende unicamente dall'ampiezza dell'intervallo di partenza e dalla precisione finale, secondo la seguente formula

$$n = \log_2 \left(\frac{b-a}{\varepsilon} \right)$$

Ad esempio, se la precisione finale da raggiungere è $\varepsilon = 1E-15$, partendo da un intervallo di ampiezza unitaria $(b-a) = 1$, occorreranno circa 50 iterazioni, per qualunque tipo di funzione. Questo fatto può essere sfruttato anche per definire un indice di efficienza relativo con cui confrontare differenti algoritmi

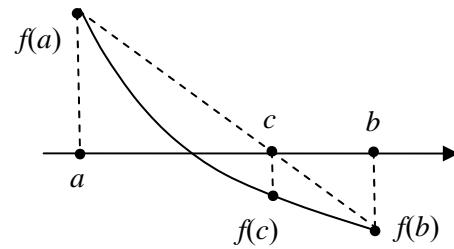
Un altro vantaggio del metodo, molto apprezzato nel calcolo automatico, è il mantenimento della segregazione della radice (bracketing). Questo significa che la radice non esce mai dall'intervallo indipendentemente dalla forma della funzione e dal numero d'iterazioni effettuate.

Metodo Regula Falsi

Antico metodo tuttora usato, specialmente nel calcolo manuale e didattico, si basa sull'interpolazione lineare della funzione $f(x)$ fra due estremi $[a, b]$ in cui la funzione a segno opposto.

Il valore c è l'incontro della retta passante per i punti $A(a, f_a)$ e $B(b, f_b)$ con l'asse x .

Se il valore $f(c) < 0$ allora la radice si trova in $[a, c]$ altrimenti si troverà nell'intervallo $[c, b]$



Il processo, analogo a quello di bisezione, può ripetersi a partire dal nuovo intervallo. Nella scelta dei nuovi intervalli $[a, b]$ deve sempre essere $f(a)f(b) < 0$. Questa opposizione dei segni è la garanzia del successo del metodo. Iterazione dopo iterazione il punto c convergerà alla vera radice. Analogamente al metodo di bisezione anche questo metodo mantiene il bracketing per cui è garantita la convergenza al 100%. Rispetto a questo però a diverse differenze che ne caratterizzano il funzionamento. Può accadere che uno degli estremi (ad esempio l'estremo "a" della figura) può rimanere inalterato per tutto il processo e per questo l'intervallo che racchiude la radice non converge necessariamente a zero. Questo va tenuto conto nella scelta dei criteri di arresto scartando il test $|b-a| < \epsilon$.

Un'altra differenza è che il numero d'iterazioni dipende dalla forma della funzione $f(x)$; questo permette al processo di raggiungere velocità migliori del metodo di bisezioni (ma anche peggiori come vedremo)

La formula interpolante per calcolare il valore c è: $c = b - \frac{f(b)}{m}$

dove: $m = \frac{f(a) - f(b)}{a - b}$ è il coefficiente angolare, o pendenza, della retta interpolante

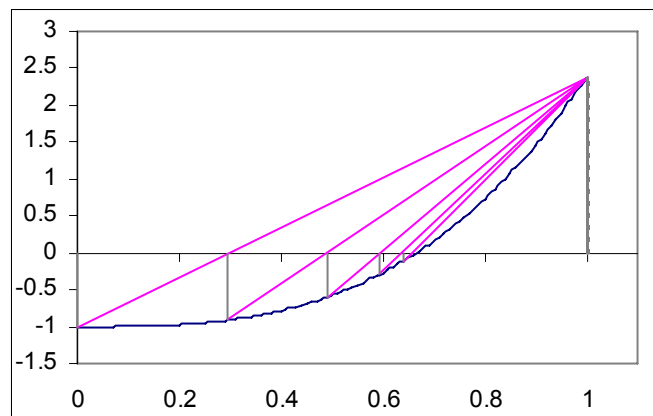
Quindi tutto il cuore dell'algorithmo può essere condensato in poche linee.

```

m = (fa-fb)/(a-b)           'calcolo pendenza
c = b-fb/m                 'calcolo nuovo punto
fc = f(c)                  'calcolo la funzione
if fc < 0 then
    b = c, fb = fc
else
    a = c, fa = fc
endif
    
```

Costo computazionale. Il costo del metodo è basso: $5 \text{ op} + 1 \text{ f}$

Vediamo come lavora questo metodo con la funzione test $f(x) = (3x/2)^3 - 1$. Assumiamo come intervallo di partenza il segmento $[0, 1]$. Il metodo converge alla radice $x = 2/3$ con un errore di circa $\epsilon \approx 3E-15$ in 35 iterazioni. La costante asintotica misurata è $c \approx 0.368$. Si osserva che l'estremo $b = 1$ rimane fisso per tutte le iterazioni. Questo succede quando la funzione è sempre concava o convessa



Compariamo il metodo "regula falsi" con il metodo di bisezione. Chiaramente il numero d'iterazione è stato sensibilmente minore (35 contro 50) ma il costo dell'algorithmo è maggiore.

Assumendo il costo del calcolo della funzione pari a 5 op (1 f = 5 op) possiamo confrontare il costo totale dei due algoritmi per questo specifico problema

Metodo	iter.	costo/iter	costo tot
Bisezione	50	7	350
Regula Falsi	35	10	350

Quindi come si vede l'efficienza globale dei due metodi è stata praticamente identica.

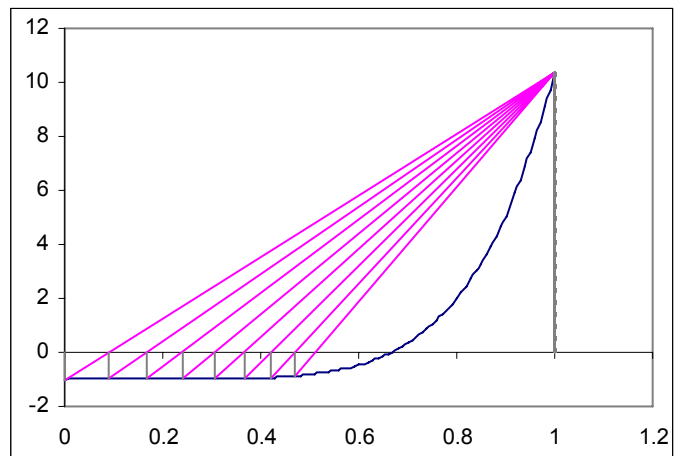
Vi sono casi però in cui l'algorithmo "Regula Falsi" va in crisi e la convergenza, sempre garantita, diventa molto lenta con velocità di convergenza inferiore all'algorithmo di bisezione Vediamo questo esempio di test.

$$f(x) = (3x/2)^6 - 1$$

Assumiamo come intervallo di partenza il segmento [0, 1]

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 3E-15$ in ben 111 iterazioni. La costante asintotica misurata è $c \cong 0.711$

Come si vede il processo di avvicinamento alla radice, pur se garantito, è molto lento, inferiore persino a quello di bisezione.



Questo comportamento si verifica quando la funzione presenta ampi tratti con pendenza quasi nulla, seguiti da brevi tratti con grande pendenza.

L'unico modo per evitare tali situazioni è quello di restringere l'intervallo il più possibile

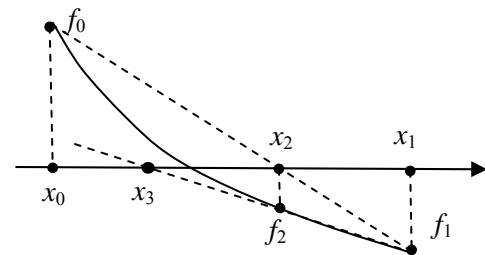
Il metodo ha convergenza lineare (ordine $p = 1$), ma la costante asintotica può variare, come abbiamo visto, secondo il comportamento locale della funzione.

Metodo della Secante

Il metodo delle secanti è una variazione della regola falsi in cui si rinuncia alla garanzia dei segni opposti. L'approssimazione della radice nell'intervallo $[x_0, x_1]$ si basa ancora sull'interpolazione lineare della funzione $f(x)$ per cui la formula è

$$x_{i+1} = x_i - \frac{f(x_i)}{m_i} \quad \text{dove} \quad m_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Il metodo richiede due valori iniziali che, in genere, sono gli estremi dell'intervallo di partenza $[a, b]$ cioè $[x_0, x_1] = [a, b]$. Il processo genera il punto x_2 . Con $[x_1, x_2]$, il processo genera x_3 ecc. In pratica ogni punto viene generato dagli ultimi due punti che, se il metodo converge, sono quelli più vicini alla radice. Questa semplice variazione dell'algoritmo (la formula è la stessa della "Regula falsi") produce una forte accelerazione della convergenza.



Per contro si perde la garanzia della convergenza e anche del bracketing.

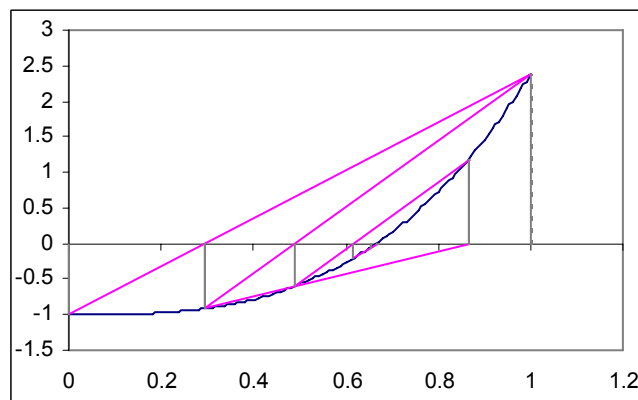
Costo computazionale. Il costo del metodo è basso: $5 \text{ op} + 1 \text{ f}$

Vediamo questo metodo con la funzione test

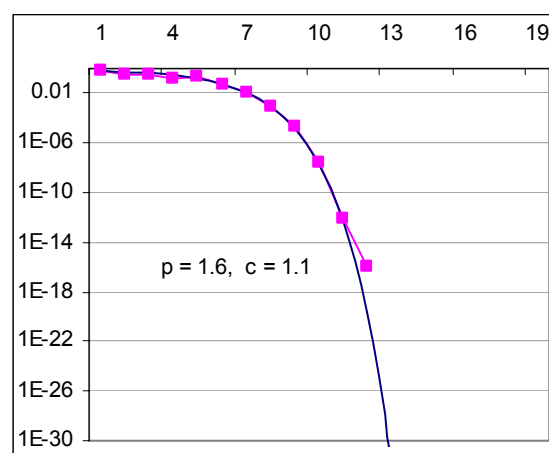
$$f(x) = (3x/2)^3 - 1$$

Assumiamo come intervallo di partenza il segmento $[0, 1]$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 11 iterazioni. L'ordine di convergenza medio è di $p = 1.6$. La costante asintotica misurata è $c \cong 1.1$. Costo tot = $11 \cdot 10 = 110$



x_i	f_i	$ e_i $
0	-1	0.6666667
1	2.375	0.3333333
0.296296296	-0.912209	0.3703704
0.491575818	-0.599091	0.1750908
0.865207005	1.185918	0.1985403
0.616975298	-0.207358	0.0496914
0.653919035	-0.056274	0.0127476
0.667679559	0.0045649	0.0010129
0.66664707	-8.82E-05	1.96E-05
0.666666637	-1.34E-07	2.975E-08
0.666666667	3.935E-12	8.744E-13
0.666666667	1.00E-16	1.00E-16



L'efficienza rispetto al metodo di bisezione in tal caso è stato di $350 / 110 \cong 3.2$, cioè più di tre volte. E' sorprendente che la stessa formula di estrapolazione usata anche nel metodo "regula falsi" possa dare risultati così differenti semplicemente adottando una diversa strategia nella scelta dei punti.

della successione $\{x_n\}$. Si osserva inoltre che non è necessario che l'intervallo di partenza $[x_0, x_1]$ contenga la radice da approssimare. Il metodo può essere avviato altrettanto bene con due punti qualsiasi purché vicini alla radice. Ripetiamo il test, ad esempio, con $[x_0, x_1] = [1, 0.8]$. La sequenza è persino più veloce, convergendo in 9 iterazioni.

Tanta efficienza deve avere un rovescio. E infatti il metodo può cadere facilmente in difficoltà per certi tipi di funzioni. Ad esempio ripetiamo il test innalzando la potenza da 3 a 4

test $f(x) = (3x/2)^4 - 1$

Intervallo di partenza $[0, 1]$

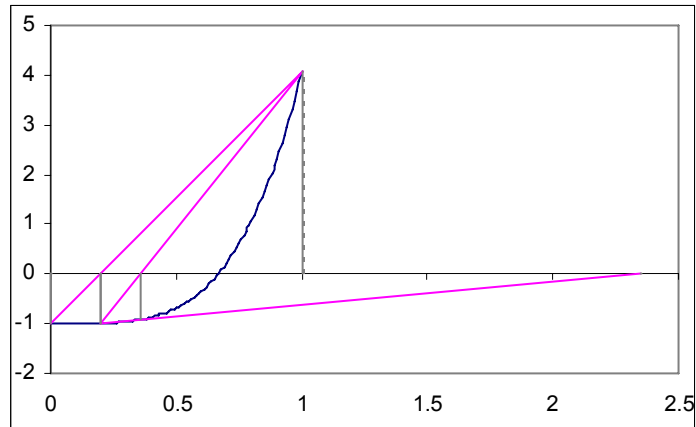
Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 18 iterazioni

L'ordine di convergenza medio è di $p = 1.6$

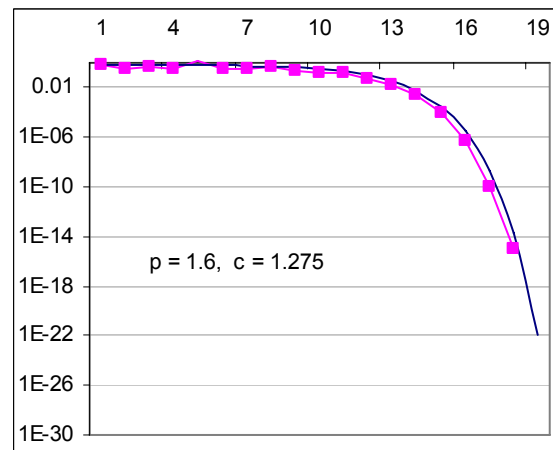
La costante asintotica misurata è $c \cong 1.275$

Costo tot = $18 \cdot 10 = 180$

Osserviamo che alcuni punti della sequenza si sono allontanati molto dalla radice



Questa instabilità è comune ai metodi molto veloci. Le prime iterazioni tendono a divergere per poi ritornare bruscamente negli ultimi passi a convergere alla radice. In pratica le prime 10 iterazioni non producono nessuna riduzione dell'errore e sono del tutto inutili ai fini del processo iterativo. In ogni caso, il metodo, pur diminuendo di efficienza resta ancora competitivo sia con il metodo di bisezione che con la "regula falsi", in quanto il costo totale è 180.



Tuttavia se ripetiamo il test innalzando la potenza da 4 a 6, $f(x) = (3x/2)^6 - 1$, il metodo diverge totalmente.

Il fatto che la successione $\{x_n\}$ tende a sfuggire dall'intervallo di partenza è certamente un fattore negativo di questo metodo. In presenza di altre radici il metodo può convergere ad una radice non voluta. In generale, nei programmi automatici, la divergenza può essere facilmente intercettata e bloccata ma la convergenza a radici non volute è molto più difficile da controllare.

E' forse questo il difetto più critico dei metodi "no-bracketing" come quello della secante e molti altri.

Metodo Secante back-step

Un modo per rendere sensibilmente più stabile il metodo della secante è la variante "back-step". Si parte da un intervallo $[a, b] = [x_0, x_1]$ contenente la radice, cioè tale che $f(x_0) \cdot f(x_1) < 0$.

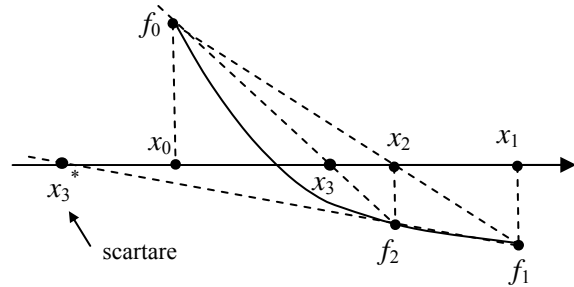
Ad ogni iterazione si controlla se $a < x_{i+1} < b$.

In caso negativo si sostituisce x_{i-1} con x_{i-2} e $f(x_{i-1})$ con $f(x_{i-2})$ e si riparte.

Nell'esempio schematizzato, il metodo produce la sequenza x_0, x_1, x_2, x_3^* . l'ultimo valore viene scartato perché esce dall'intervallo iniziale.

Il valore x_1 viene sostituito con x_0 e si riparte con la sequenza x_0, x_2 , che produce x_3 , e così via.

In pratica si vuole evitare che i valori della sequenza si allontanino dall'intervallo iniziale.



Nei casi "difficili" la variante back-step della secante si comporta come la "regola falsi"; nei casi favorevoli acquista la velocità della secante. Pur non essendo garantita la convergenza questo metodo risulta più robusto di quello della secante pura conservando al tempo stesso una buona efficienza in prossimità della radice.

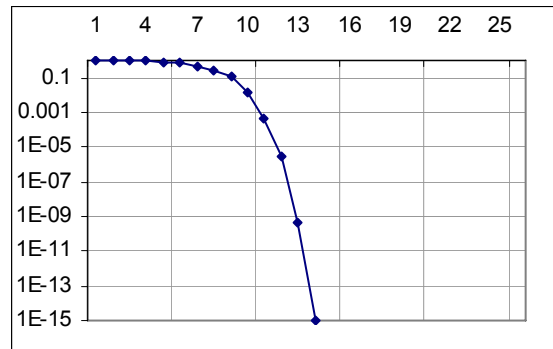
La verifica della condizione $a < x_{i+1} < b$ ad ogni iterazione comporta 2 operazioni aggiuntive per cui il costo è lievemente superiore.

Costo computazionale. Il costo del metodo è basso: $7 \text{ op} + 1 \text{ f}$

test $f(x) = (3x/2)^4 - 1$

Intervallo di partenza $[0, 1]$

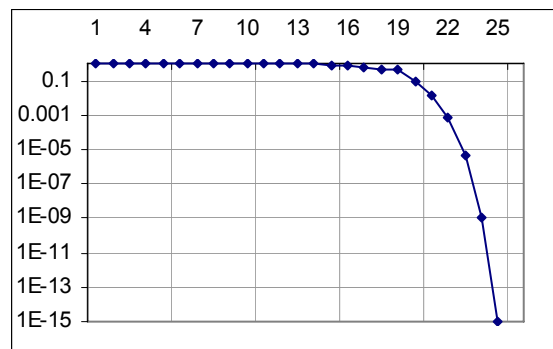
Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 13 iterazioni
L'ordine di convergenza medio è di $p = 1.6$
La costante asintotica misurata è $c \cong 1.8$
Costo tot = $13 \cdot 12 = 156$



test $f(x) = (3x/2)^6 - 1$

Intervallo di partenza $[0, 1]$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 24 iterazioni
Costo tot = $24 \cdot 12 = 288$
Notare che questo test non viene superato dal metodo della secante



Metodo Pegasus

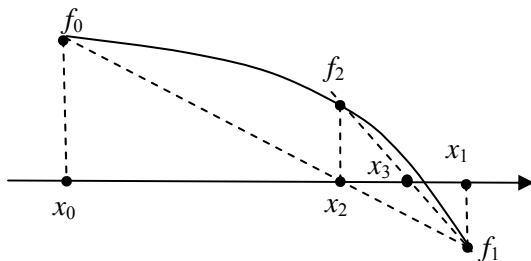
I tentativi di conciliare la robustezza del metodo "regula falsi" con la velocità del metodo della secante hanno condotto a diversi ingegnosi algoritmi. Uno dei più interessanti è il cosiddetto "Pegasus"¹.

Il metodo viene innescato con i due estremi dell' intervallo iniziale in cui è verificata la condizione di opposizione dei segni $f(x_0) \cdot f(x_1) < 0$. L'approssimazione della radice nell'intervallo $[x_0, x_1]$ si basa ancora sull'interpolazione lineare della funzione $f(x)$ per cui la formula è ancora

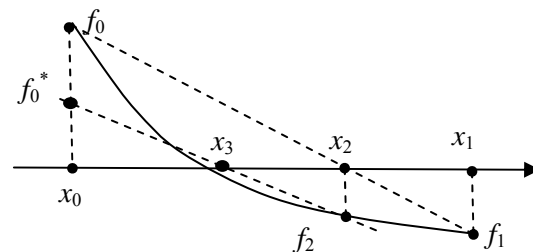
$$x_{i+1} = x_i - \frac{f(x_i)}{m_i} \quad \text{dove} \quad m_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Per avere una velocità di convergenza della secante il metodo usa gli ultimi due valori della successione, che sono presumibilmente i più vicini alla radice. Inoltre, per conservare la garanzia del bracketing viene effettuato il test dei segni ad ogni iterazione.

Si possono verificare due casi:



Gli ultimi due valori della successione hanno lo segno differente $f_1 f_2 < 0$
 Tutto bene. La radice si trova fra x_1 e x_2
 Si calcola l'interpolazione x_3 fra i punti (x_1, f_1) e (x_2, f_2)



Gli ultimi due valori della successione hanno lo stesso segno $f_1 f_2 > 0$
 Si riduce il valore di f_0 del fattore $k = f_1 / (f_1 + f_2)$
 si pone $f_0^* = k f_0$ e si calcola l'interpolazione x_3 fra i punti (x_1, f_1) e (x_0, f_0^*)

Si osserva che il secondo caso, $f_1 f_2 > 0$, è il più critico perché può far uscire la successione dall' intervallo iniziale. Questo succede ad esempio quando i due valori f_1 e f_2 sono quasi simili e la retta interpolante diventa quasi parallela all'asse delle x.

Il fattore $k = f_1 / (f_1 + f_2)$ è sempre positivo e minore di 1; di conseguenza il punto (x_0, f_0^*) giace sempre sul segmento (x_0, f_0) e quindi il punto d'interpolazione x_3 è sempre compreso fra x_0 e x_2 , all'interno dell'intervallo di partenza. Questo geniale accorgimento assicura il bracketing e quindi la convergenza del metodo al 100%. Inoltre si osserva che nel caso critico $f_1 \cong f_2$, il fattore di riduzione k tende a 0.5; il valore f_0^* tende a $f_0/2$ e il punto x_3 si avvicina ancor più alla radice, come mostrato in figura.

Costo computazionale: il calcolo del valore ridotto f_0^* necessita di 3 operazioni aggiuntive rispetto all'algoritmo della secante. Tuttavia il calcolo viene effettuato solo quando si verifica la condizione di uguaglianza di segni a cui, per le regole del costo, attribuiamo il 50% di probabilità. Quindi il costo dell'algoritmo Pegasus è: $5 + 0.5 \cdot 3 \text{ op} + 1 f = 6.5 \text{ op} + 1 f$

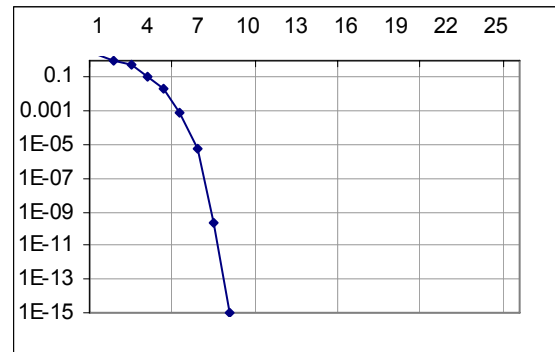
Vediamo il suo comportamento con le funzioni test

¹ Proposto nel 1971 da M. Dowell and P. Jarratt

test $f(x) = (3x/2)^3 - 1$

Intervallo di partenza $[0, 1]$

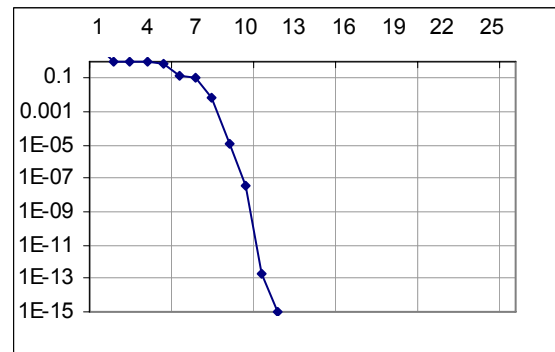
Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 9 iterazioni
 L'ordine di convergenza medio è di $p = 1.6$
 La costante asintotica misurata è $c \cong 0.8$
 Costo tot = $9 \cdot 11.5 = 103.5$



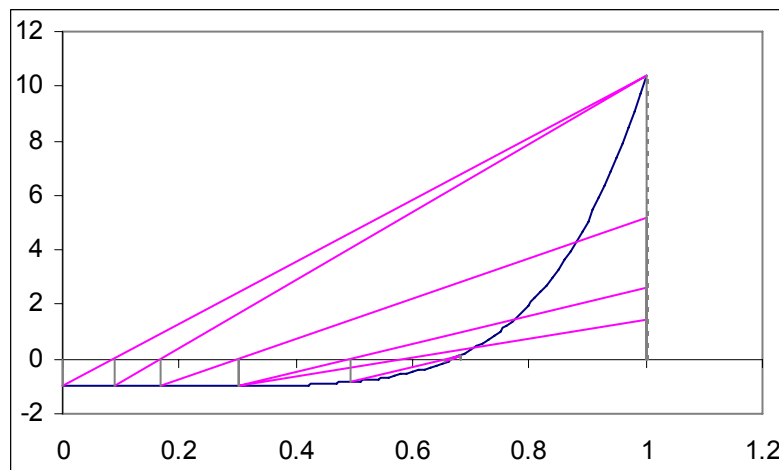
test $f(x) = (3x/2)^6 - 1$

Intervallo di partenza $[0, 1]$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 12 iterazioni
 L'ordine di convergenza medio è di $p = 1.6$
 La costante asintotica misurata è $c \cong 1.2$
 Costo tot = $12 \cdot 11.5 = 138$



In quest'ultimo caso di test il metodo della secante fallisce mentre il metodo regula falsi impiega più di 100 iterazioni. E' interessante osservare sul grafico le prime iterazioni del metodo per capire come raggiunge la radice senza cadere nelle trappole degli altri due metodi: bassa convergenza e divergenza.



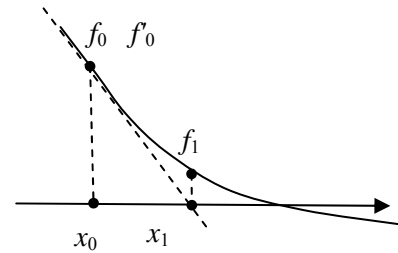
Come si nota chiaramente, per quattro iterazioni il punto $b = 1$ non viene mai cambiato ma viene ridotto di volta in volta il valore della funzione proporzionalmente al fattore k .

Le caratteristiche di questo metodo si sono rivelate globalmente ottime in molti altri casi di test: alta affidabilità, eccezionale robustezza, velocità, efficienza e assenza del calcolo delle derivate. Possiamo dire, dopo lunghi test, che l'unica criticità rilevata sono le radici multiple, cosa del resto comune a molti altri metodi di ordine più elevato.

Metodo Newton-Raphson

Si tratta di uno dei più popolari ed efficienti metodi usati per la risoluzione di equazioni non lineari, noto anche come metodo delle tangenti. La sua semplicità concettuale e geometrica lo rende particolarmente attraente sia dal punto di vista della didattica che della teoria. Questo metodo può essere generalizzato anche al caso di più variabili e alle radici complesse. In questo capitolo verrà trattato il caso delle funzioni con una variabile reale.

Il metodo di Newton-Raphson-Fourier si può adottare nell'ipotesi che $f(x)$ sia derivabile ed ammetta derivata prima continua. Nella figura è riportata l'interpretazione geometrica di tale metodo. Da punto iniziale x_0 , si calcolano la funzione f_0 e al sua derivata prima f'_0 , e si considera la retta tangente passante per il punto (x_0, f_0) . L'intersezione di tale retta con l'asse x dà il nuovo punto x_1 .



L'equazione della retta è data da

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

da cui, essendo $f(x_1) = 0$, si ha

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0 \Rightarrow x_1 - x_0 = -f(x_0) / f'(x_0)$$

Il processo può essere ripetuto dando così il metodo iterativo

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1}$$

Per il metodo di Newton è possibile fornire la formula iterativa dell'errore

Prendiamo il polinomio di Taylor di 2° grado per la funzione $f(x)$ a partire dal punto x_i

$$f(x) \cong f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2 \tag{2}$$

Per $x = \tilde{x} \Rightarrow f(\tilde{x}) = 0$

$$f(x_i) + f'(x_i)(\tilde{x} - x_i) + \frac{1}{2} f''(x_i)(\tilde{x} - x_i)^2 \cong 0 \tag{3}$$

Dalla (1) si ha

$$f(x_i) = -f'(x_i)(x_{i+1} - x_i) \tag{4}$$

da cui sostituendo (4) in (3)

$$\tilde{x} - x_{i+1} \cong -\frac{f''(x_i)}{2f'(x_i)} (\tilde{x} - x_i)^2$$

Posto $\tilde{x} - x_i = e_i$, prendendo i valori assoluti e passando al limite $x_i \rightarrow \tilde{x}$ si ha

$$|e_{i+1}| \cong \left| \frac{f''(x_i)}{2f'(x_i)} \right| |e_i|^2 \Rightarrow \lim_{i \rightarrow \infty} \frac{|e_{i+1}|}{|e_i|^2} = \left| \frac{f''(\tilde{x})}{2f'(\tilde{x})} \right| \tag{5}$$

L'ultima relazione ci dice che per $f'(\tilde{x}) \neq 0$, cioè per radice \tilde{x} semplice, l'ordine di convergenza del metodo di Newton è almeno 2. Se anche $f''(\tilde{x}) \neq 0$ allora il limite è finito e diverso da zero;

quindi il metodo ha esattamente ordine $p = 2$ e costante asintotica $c = c = |f''(\tilde{x})| / |2f'(\tilde{x})|$

Questo significa che il numero di cifre esatte raddoppia approssimativamente ad ogni iterazione.

La relazione precedente vale nell'ipotesi che la radice sia semplice

Se la radice ha molteplicità $m > 1$ allora si può esprimere la funzione $f(x)$ in un conveniente intorno della radice \tilde{x} come

$$f(x) \cong a(x - \tilde{x})^m \tag{6}$$

Derivando

$$f'(x) \cong m \cdot a(x - \tilde{x})^{m-1} \tag{7}$$

La funzione iterativa di Newton è

$$\phi(x) = x - \frac{f(x)}{f'(x)} \tag{8}$$

da cui sostituendo

$$\phi(x) = x - \frac{a(x - \tilde{x})^m}{m \cdot a(x - \tilde{x})^{m-1}} \Rightarrow \phi(x) = x - \frac{(x - \tilde{x})}{m}$$

e derivando

$$\phi'(x) = 1 - 1/m \tag{9}$$

Questa relazione ci dice che per $m > 1$ si ha $0 < \phi'(x) < 1$. Quindi il metodo converge linearmente con costante asintotica $c = (m-1)/m$

Costo computazionale: Il costo del metodo dipende molto dalla derivata della funzione. Comunque per funzioni che s'incontrano comunemente il costo è basso: $2 \text{ op} + 2 \text{ f}$

Nel caso che la derivata venga approssimata mediante le differenze divise il costo sale a $2 \text{ op} + 3 \text{ f}$

Vediamo il suo comportamento con le funzioni test

test $f(x) = (3x/2)^3 - 1$

Punto d'innescio $x_0 = 1$

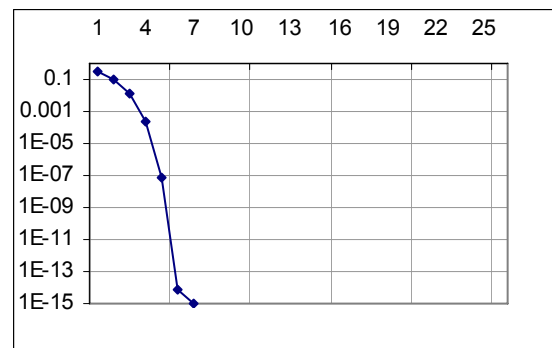
Il metodo converge alla radice $x = 2/3$ con un

errore di circa $e \cong 1E-15$ in 6 iterazioni

L'ordine di convergenza medio è di $p = 2$

La costante asintotica misurata è $c \cong 1$

Costo tot = $6 \cdot 12 = 72$



test $f(x) = (3x/2)^6 - 1$

Punto d'innescio $x_0 = 1$

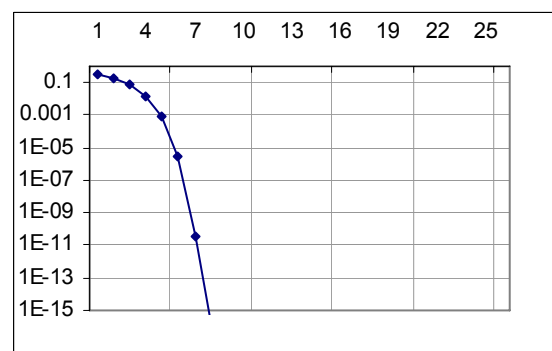
Il metodo converge alla radice $x = 2/3$ con un

errore di circa $e \cong 1E-15$ in 7 iterazioni

L'ordine di convergenza medio è di $p = 1.9$

La costante asintotica misurata è $c \cong 1.8$

Costo tot = $7 \cdot 12 = 84$



Notare che, anche con il calcolo di due funzioni per iterazione, l'efficienza rimane molto alta grazie sia alla velocità e sia al basso costo computazionale. Inoltre il numero d'iterazioni varia molto poco

per entrambi i casi di test. Si deve ricordare che, nel secondo test, il metodo Regula falsi ha richiesto più di 100 iterazioni.

A differenza dei metodi basati sulla scelta di un intervallo contenente la radice, il metodo delle tangenti viene innescato con un solo punto. In particolare non avremo a priori nessun controllo sulla posizione del punto successivo. I punti possono allontanarsi, oscillare, divergere o incontrare un punto singolare dove la derivata si annulla. Ad esempio nei due test precedenti il punto $x_0 = 0$ non può essere usato per l'innescio.

Questo suggerisce che per sfruttare la sua eccellente velocità l'algoritmo di Newton necessita di una prima approssimazione ragionevolmente accurata e che il suo compito sia quello di affinare l'approssimazione.

La relazione iterativa dell'errore (5) garantisce la convergenza "locale" del metodo, cioè in un intorno sufficientemente piccolo della radice.

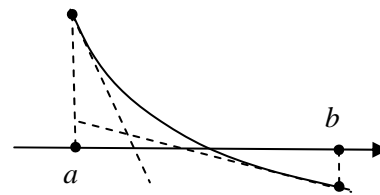
Esistono però delle condizioni cosiddette "globali". Se $f(x) \in C^2$ nell'intervallo $[a, b]$ e se si verifica

1. $f(a)f(b) < 0$ signi opposti agli estremi
2. $f'(x) \neq 0 \quad \forall x \in [a, b]$ derivata prima non nulla in $[a, b]$
3. $f''(x) > 0 \vee f''(x) < 0 \quad \forall x \in [a, b]$ derivata seconda di segno costante in $[a, b]$
4. $|f(a)| < |f'(a)|(b-a) \wedge |f(b)| < |f'(b)|(b-a)$

Allora il metodo di Newton converge alla radice per ogni punto d'innescio $\forall x_0 \in [a, b]$

Geometricamente parlando la condizione 3) significa che la curva è tutta concava o convessa.

La condizione 4) significa che le tangenti agli estremi dell'intervallo devono intersecare l'asse delle ascisse all'interno di $[a, b]$



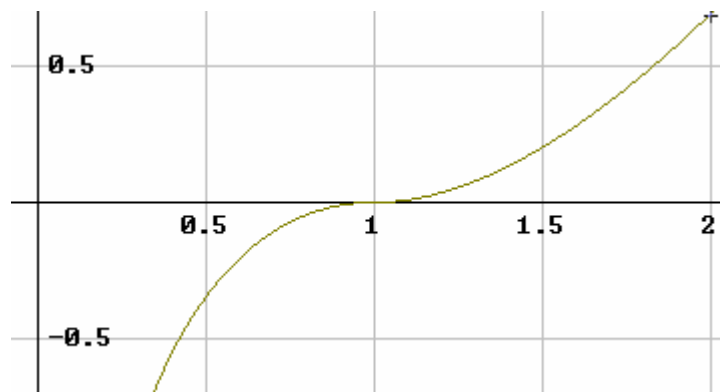
Radici multiple

Applichiamo il metodo di Newton alla funzione test $f(x) = |x-1| \log(x)$ che ha uno zero per $x = 1$.

la derivata della funzione è

$$f'(x) = \text{sgn}(x-1) \left(\log(x) + \frac{x-1}{x} \right)$$

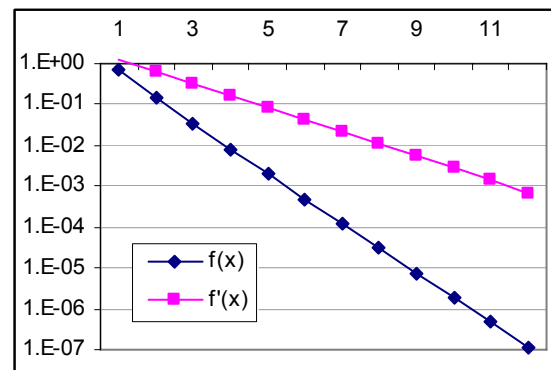
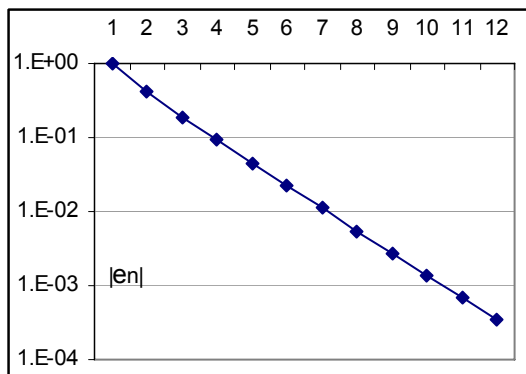
Scegliamo come punto di partenza $x_0 = 2$



x	$f(x)$	$f'(x)$	dx
2	0.693147181	1.193147181	0.580940216
1.419059784	0.146668632	0.645302583	0.227286602
1.191773182	0.033645122	0.336356428	0.100028181
1.091745001	0.008053132	0.171812526	0.046871621
1.04487338	0.001969749	0.086841947	0.022681998

1.022191382	0.000487073	0.043658352	0.011156463
1.011034919	0.000121102	0.021888957	0.005532583
1.005502335	3.01927E-05	0.010959478	0.00275494
1.002747395	7.53783E-06	0.005483496	0.00137464
1.001372756	1.88317E-06	0.002742688	0.000686613

Come si vede, la convergenza alla radice 1 è evidente ma lenta; dopo 10 iterazioni la precisione della radice è circa 1E-3. Si osserva che la funzione $f(x)$ tende molto più rapidamente a zero di $\{e_n\}$. I grafici seguenti mostrano le curve delle successioni $\{e_n\}$, $\{|f(x_n)|\}$, $\{|f'(x_n)|\}$. Le traiettorie appaiono chiaramente lineari. Inoltre notiamo che sia la funzione, sia la derivata tendono a zero, con differenti costanti asintotiche. Questi indizi rivelano che la radice ha molteplicità > 1



Come accelerare la convergenza a radici multiple

Si può verificare che il metodo Δ^2 non funziona per questa successione.

Tuttavia la relazione iterativa (8) ci fornisce la chiave per accelerare la convergenza

Infatti se consideriamo la relazione iterativa modificata

$$\phi(x) = x - \alpha \frac{f(x)}{f'(x)}$$

dove α un opportuno coefficiente. Sostituendo (6) e (7) e derivando si ha

$$\phi(x) = x - \alpha \frac{(x - \tilde{x})}{m} \Rightarrow \phi'(x) = 1 - \frac{\alpha}{m}$$

Per la condizione di convergenza

$$0 \leq \phi'(x) < 1 \Rightarrow 0 < \alpha \leq m$$

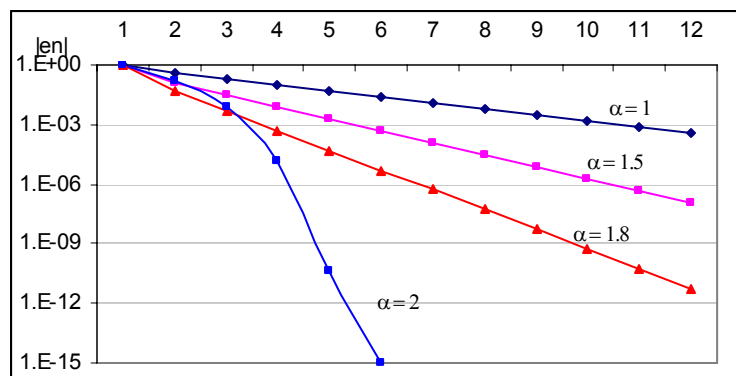
Il parametro α può essere scelto a piacere: se $\alpha = m$ allora $\phi' = 0$ e quindi il metodo Newton modificato ha ancora ordine > 1 ; se $\alpha = 1$ ritorna il metodo di Newton originale.

Tutti i valori intermedi $1 < \alpha \leq m$ permettono di modulare la convergenza.

Nella tabella a fianco sono state tracciate le traiettorie dell'errore del metodo Newton accelerato

$$x_{i+1} = x_i - \alpha \cdot f(x_i) / f'(x_i)$$

applicato alla funzione test precedente.

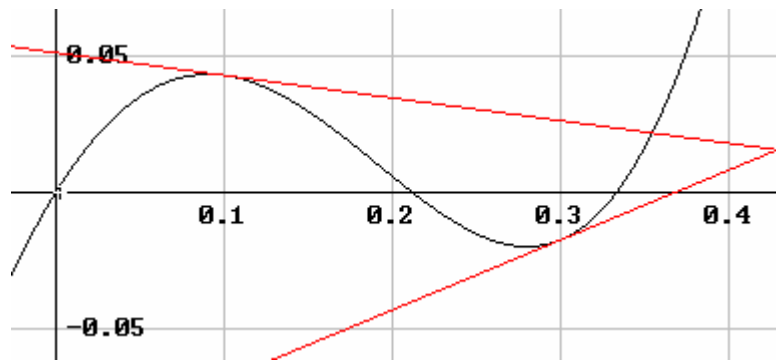


Come già anticipato il difetto peggiore dei metodi cosiddetti "liberi", cioè non vincolati in un intervallo, è quello di convergere a radici non volute.

Esempio: $f(x) = x - 6x^2 \cos(\pi x)$ che ha uno zero nell'intervallo $[0.1, 0.3]$ dato da $x \cong 0.2119$.

Il metodo converge però alla radice $x \cong 0.3333$ sia che si parta da $x_0 = 0.1$, sia che si parta da 0.3

Partendo da un punto più vicino alla radice come $x_0 = 0.2$, si ha invece la convergenza alla radice voluta



Al contrario un qualunque metodo di bracketing non ha nessuna difficoltà a trovare la radice voluta.

Negli algoritmi ad un punto innesco come quello di Newton, viene sempre raccomandata, nei libri, la scelta del punto "sufficientemente vicino" alla radice. Un modo elegante e rapido per eludere la questione. Il concetto di "sufficientemente vicino" dal punto di vista numerico è molto approssimativo. Vediamo un esempio.

Riprendiamo la funzione analizzata nel capitolo "Separazione delle radici" $f(x) = x^3 - \log(x) - 10$.

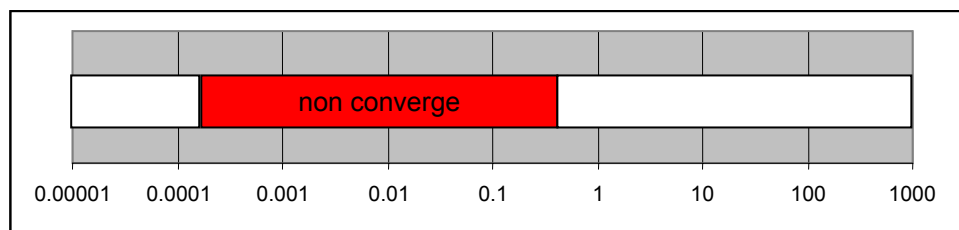
Come è stato mostrato la funzione ha due zeri positivi: uno vicino a 2 e uno vicino a 0 (quest'ultimo molto "nascosto").

La derivata della funzione è $f'(x) = 3x^2 - 1/x$. Innescando il metodo di Newton con $x_0 = 10$ il metodo converge in circa 10 iterazioni alla radice $x_{10} \cong 2.2099388$ con $1E-15$ di precisione.

Ripetendo il processo partendo da un valore molto lontano, $x_0 = 1000$, il metodo converge alla stessa radice in circa 21 iterazioni. Proviamo ora a cercare l'altra radice, quella vicino alla zero.

Partiamo da valori positivi molto vicino allo zero $x_0 = 0.1$, Poiché il metodo fallisce, prendiamo un valore ancora più vicino $x_0 = 0.01$ e poi $x_0 = 0.001$. Il metodo continua fallire!

Inaspettatamente scendendo al valore $x_0 = 0.0001$ il metodo converge rapidamente alla radice cercata $x_7 \cong 4.53999E-05$. Gli insiemi di convergenza sono schematizzati nelle seguente grafico



Quindi come si vede, il concetto di "sufficientemente vicino" è ambiguo. Per alcune radici la "vicinanza" si estende a valori molto grandi mentre per altre nemmeno $1E-3$ può essere sufficiente. Il concetto "sufficientemente vicino" dovrebbe piuttosto essere sostituito con quello geometrico di "curva sufficientemente rettilinea". Se fra il punto d'innescio e la radice la curva è abbastanza rettilinea allora il metodo converge, indipendentemente dalla distanza reciproca.

Inoltre per alcune funzioni "strane" come $f(x) = \text{sgn}(x) \cdot \sqrt[3]{x^2}$ il metodo di Newton non converge per alcun punto d'innescio nell'intorno di $x = 0$.

Metodo Halley

Si tratta di un metodo del 3 ordine ad altissima velocità di convergenza, conosciuto come il metodo delle tangenti iperboliche o formula di Halley. Usa sia la derivata prima che la derivata seconda.

Può essere ricavato in vari modi. Il più immediato è l'espansione polinomiale di Taylor di 2° grado

$$f(x) \cong f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2$$

L'intersezione x_{i+1} con l'asse x si trova imponendo $f(x_{i+1}) = 0$

$$f'(x_i)(x_{i+1} - x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i)^2 = -f(x_i)$$

Ovvero

$$(x_{i+1} - x_i) \left(f'(x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i) \right) = -f(x_i)$$

$$x_{i+1} - x_i = \frac{-f(x_i)}{f'(x_i) + \frac{1}{2} f''(x_i)(x_{i+1} - x_i)}$$

L'incremento $(x_{i+1} - x_i)$ nell'espressione a destra può essere approssimato con la formula di Newton

$$x_{i+1} - x_i = -f(x_i) / f'(x_i)$$

Da cui sostituendo

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - \frac{1}{2} f(x_i)f''(x_i)} \quad (1)$$

Si può dimostrare che, per radici semplici, il metodo converge con ordine cubico con costante asintotica.

$$c = - \left\{ \frac{f'''(\tilde{x})}{f'(\tilde{x})} - \frac{3}{2} \left[\frac{f''(\tilde{x})}{f'(\tilde{x})} \right]^2 \right\}$$

Costo computazionale: Il costo del metodo di Halley è determinato in massima parte dal calcolo delle derivate prima e seconda. Per funzioni comuni possiamo stimare il costo delle derivate equivalente a quello della funzione stessa, per cui il costo per iterazione è medio-alto: 7 op è 3 f. Nel caso che di approssimazione mediante le differenze divise il costo sale a: 7 op + 5 f

In letteratura si trovano anche altre forme del metodo di Halley

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \left(1 - \frac{f(x_i)f''(x_i)}{2[f'(x_i)]^2} \right)^{-1} \quad (2)$$

$$x_{i+1} = x_i - \left(\frac{f'(x_i)}{f(x_i)} - \frac{f''(x_i)}{2f'(x_i)} \right)^{-1} \quad (3)$$

L'ultima formula è quella con costo minimo: 6 op + 3 f. Tuttavia, in certe situazioni può portare all'overflow e si è rilevata meno stabile della (1) a causa del termine $f(x_i)$ a denominatore, che ovviamente tende a 0.

test $f(x) = (3x/2)^3 - 1$

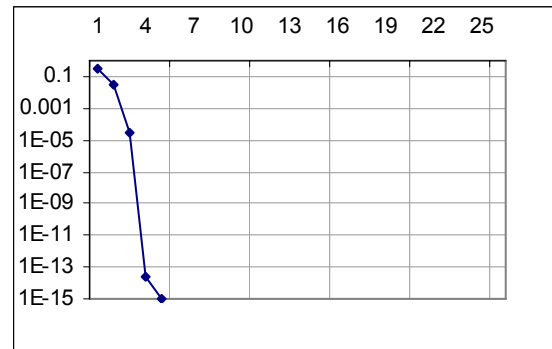
Punto d'innescio $x_0 = 1$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \approx 1E-15$ in 4 iterazioni

L'ordine di convergenza medio è di $p = 3$

La costante asintotica misurata è $c \approx 0.9$

Costo tot = $4 \cdot 21 = 84$



Sorprendentemente se ripetiamo il test per la funzione più difficile $f(x) = (3x/2)^6 - 1$ otteniamo il medesimo risultato. Chiaramente il costo del metodo è stato ben speso!

Anche questo metodo, come quello di Newton viene innescato con un punto solo e di conseguenza non mantiene il bracketing. Il suo ruolo è soprattutto quello di rifinitore di alta velocità quando si lavora in multiprecisione.

Un esempio di applicazione di questa formula è il calcolo iterativo della radice n-esima. Similmente a quanto fatto nel capitolo del metodo di Newton applichiamo la formula di Halley alla risoluzione dell'equazione $f(x) = x^n - a$. Differenziando e semplificando si ha la formula iterativa

$$x_{i+1} = 2 \frac{x(a - x^n)}{(n+1)x^n + (n-1)a}$$

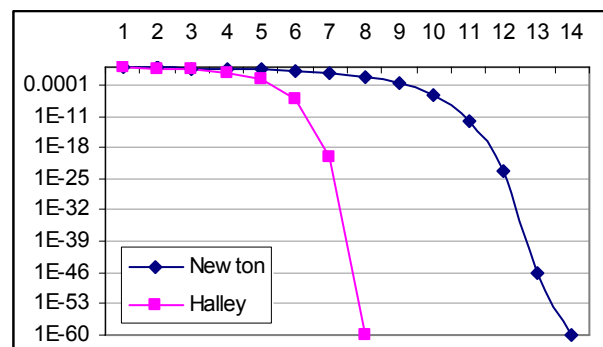
Usiamola per approssimare le prime 60 cifre¹ di $x = \sqrt[10]{2}$

Inneschiamo il processo con $x_0 = 2$

x_i	$ e_i $
2	0.92822654
1.63765289842226555575252614784612657330260592093600425456479	0.56587944
1.34761069208489276132292326445028253682493461960422896479022	0.27583723
1.14424494459639210087219464080479337411068207087652597308722	0.07247148
1.07415271928982179496187822026995003166861093552432629425482	0.00237926
1.07177355894273431344795954579371145388898984521672957239773	9.6406E-08
1.07177346253629316421944158422490505904650385822024707128142	6.4353E-21
1.07177346253629316421300632502334202290638460497755678348278	1E-60

Si noti l'impressionante accelerazione della convergenza!. Il grosso del lavoro è stato fatto praticamente nelle ultime 2-3 iterazioni.

Nel grafico a fianco sono mostrate per confronto le traiettorie dell'errore con il metodo di Halley e con il metodo di Newton. Notare come per entrambi i metodi più della metà del lavoro viene speso nell'avvicinamento alla radice (tratto quasi piatto delle curve). Arrivati nell'intorno del 10 % del valore della radice iniziano la discesa che porta la precisione ai valori di 1E-60 in pochi passi.

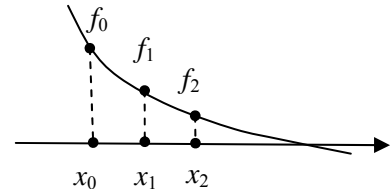


¹ Per questo calcolo è stato usato l'addin Xnumber.xla per MS Excel

Metodo Halley con differenze finite

La convergenza cubica del metodo delle tangenti iperboliche è molto attraente. Tuttavia in molti casi il calcolo della derivata prima e seconda può essere problematico. La derivazione approssimata risolve in parte questi problemi al prezzo, però, di un sensibile costo aggiuntivo (7 op + 5 f = 32 op) Per ovviare a questi problemi è stato sviluppato un metodo, derivato dalla formula di Halley, che necessita solo di un solo calcolo della funzione f(x) per ogni iterazione.

Il metodo viene innescato con tre punti $[x_0, x_1, x_2]$. Non è necessario che la radice sia contenuta nell'intervallo $[x_0, x_2]$ perché questo metodo non conserva il bracketing. Però per comodità vengono usati gli estremi dell'intervallo $[a, b]$ della radice e il suo punto medio $x_1 = (a + b)/2$. Vengono calcolate quindi le differenze finite.



$$d_1 = \frac{f_1 - f_0}{x_1 - x_0} \quad , \quad d_2 = \frac{f_2 - f_1}{x_2 - x_1} \quad , \quad d_{21} = \frac{d_2 - d_1}{x_2 - x_0}$$

Il nuovo punto x_3 viene calcolato con la formula

$$x_3 = x_2 - \frac{f_2 \cdot d_2}{d_2^2 - f_1 \cdot d_{21}} \tag{1}$$

Riapplicando il processo alla sequenza $[x_1, x_2, x_3]$ si genera x_4 e così iterando.

Quando $|x_2 - x_0| \ll 1$ le differenze finite approssimano le derivate

$$d_2 \cong f' \quad d_{21} \cong \frac{1}{2} f'' \tag{2}$$

e si può mostrare che il metodo (1) approssima la formula di Halley. Infatti sostituendo le (2) in (1) si ha

$$x_3 = x_2 - \frac{f \cdot f'}{(f')^2 - \frac{1}{2} f \cdot f''}$$

Ovviamente, a causa delle drastiche semplificazioni che abbiamo adottato il metodo non avrà convergenza come quella della formula di Halley ma sensibilmente minore. Ci attendiamo tuttavia che sia almeno significativo per un metodo senza derivate ($1 < p < 2$).

Costo computazionale. Ad ogni nuova iterazione il valore d_1 è quello di d_2 calcolato nell'iterazione precedente e quindi non viene conteggiato. Pertanto il costo è $12 \text{ op} + 1 f$

test $f(x) = (3x/2)^3 - 1$

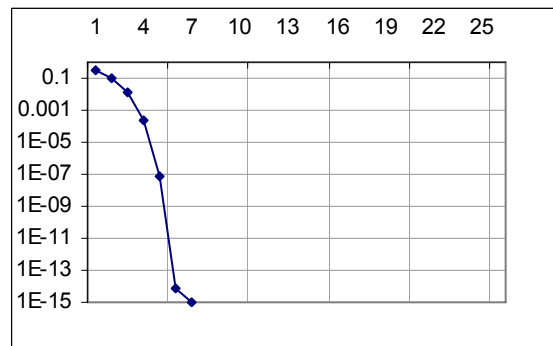
Punto d'innescio $x_0 = 1$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 6 iterazioni

L'ordine di convergenza medio è di $p = 1.9$

La costante asintotica misurata è $c \cong 1$

Costo tot = $6 \cdot 17 = 102$



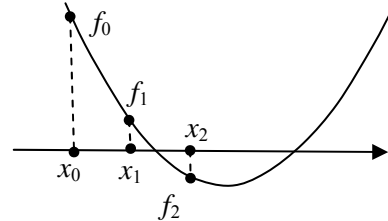
Metodo della Parabola

Si tratta di un metodo relativamente antico ottenuto estendendo il concetto d'interpolazione da quella lineare per due punti a quella parabolica per tre punti. Anche in questo caso gli algoritmi derivati sono molti (Muller, Brent, Traub, ecc.) che si differenziano per qualche accorgimento nella scelta dei punti ovvero per la famiglia di parabole usate (ad asse verticale od orizzontale)

In questo paragrafo tratteremo del metodo della parabola ad asse verticale

Dati tre punti (x_0, f_0) , (x_1, f_1) , (x_2, f_2) determiniamo l'intersezione con l'asse x della parabola ad asse verticale che interpola i punti dati. Osserviamo che l'intersezione può anche non esistere. In tal caso vedremo dopo come l'algoritmo si modifica.

Per comodità facciamo il cambio di variabile $t = x - x_2$



In questo modo il punto di zero è quello più vicino a x_2 , cioè con t minimo

Sotto queste ipotesi, la parabola generica sarà $y = at^2 + bt + c$. I coefficienti vengono determinati dai vincoli sui punti

$$y(0) = c \Rightarrow c = y_2$$

$$y_0 = at_0^2 + bt_0 + y_2 \quad \text{dove } t_0 = x_0 - x_2 \quad (1)$$

$$y_1 = at_1^2 + bt_1 + y_2 \quad \text{dove } t_1 = x_1 - x_2 \quad (2)$$

Facendo il sistema fra (1) e (2) si ha

$$\begin{cases} at_0 + b = (y_0 - y_2)/t_0 \\ at_1 + b = (y_1 - y_2)/t_1 \end{cases} \Rightarrow \begin{bmatrix} t_0 & 1 \\ t_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \end{bmatrix} \quad (3)$$

dove si è posto $d_0 = (y_0 - y_2)/(x_0 - x_2)$ e $d_1 = (y_1 - y_2)/(x_1 - x_2)$

La soluzione del sistema (3) dà:

$$a = (d_1 - d_0)/(x_1 - x_0)$$

$$b = [(x_1 - x_2)d_0 + (x_2 - x_0)d_1]/(x_1 - x_0)$$

Contando una sola volta $(x_2 - x_0)$, $(x_1 - x_2)$, $(x_1 - x_0)$, il calcolo dei coefficienti della parabola interpolante costa 13 operazioni.

Se si verifica che $\Delta = b^2 - 4ac > 0$ allora la parabola intercetta l'asse x in due punti di cui il più piccolo è dato da

$$t^* = \frac{-2c}{b + \text{sgn}(b) \cdot \sqrt{\Delta}} \Rightarrow x_3 = x_2 + t^*$$

Se si verifica che $\Delta = b^2 - 4ac \leq 0$ allora si pone $\Delta = 0$ e quindi $t^* = -2c/b$

Il valore t^* è appunto l'incremento che si deve dare a x_2 per approssimare lo zero

La funzione viene calcolata nel punto $f_3 = f(x_3)$ e il processo viene iterato con i tre punti più recenti (x_1, f_1) , (x_2, f_2) , (x_3, f_3) .

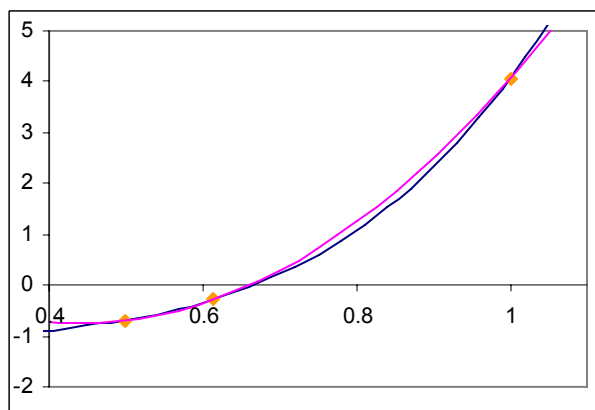
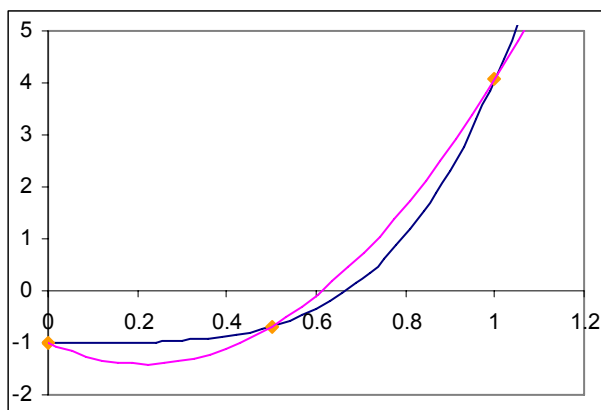
Costo computazionale: ovviamente, è molto più alto di quello lineare: $22 \text{ op} + 1 \text{ f}$

Tuttavia ci attendiamo che la convergenza del metodo sia tale da compensare il costo elevato.

Vediamo come il metodo lavora applicato alla funzione test $f(x) = (3x/2)^4 - 1$ con tripletta d'innescio $[x_0 = 0, x_1 = 0.5, x_2 = 1]$ (di solito il punto x_1 è il punto medio dell'intervallo $[x_0, x_2]$) Per ogni tripletta, vengono calcolati i coefficienti a, b, c della parabola interpolante e quindi l'incremento t . La tabella seguente riporta il calcolo dei primo 8 passi.

i	x_i	f_i	a	b	c	Δ	t
0	0	-1					
1	1	4.0625					
2	0.5	-0.68359	8.859375	5.0625	-0.68359	49.85376	0.112774292
3	0.612774292	-0.28622	15.41356	5.261914	-0.28622	45.33413	0.04772254
4	0.660496832	-0.03651	10.64704	5.740581	-0.03651	34.50909	0.006286386
5	0.666783218	0.000699	12.70728	5.998667	0.000699	35.94845	-0.00011664
6	0.666666581	-5.1E-07	13.41847	6.000007	-5.1E-07	36.00012	8.52978E-08
7	0.666666667	-8.3E-13	13.50154	6	-8.3E-13	36	1.37668E-13
8	0.666666667	1.00E-15	13.45868	6	1E-15	36	0

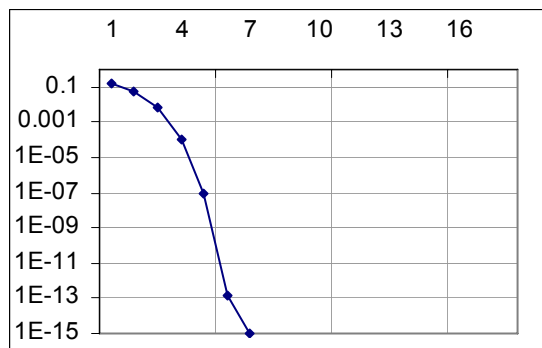
Nei grafici seguenti sono state disegnate le prime due parabole



La prima è stata ottenuta con la terna $[0, 0.5, 1]$, al seconda con $[1, 0.5, 0.6127]$. Come si vede le successive approssimazioni convergono molto velocemente alla radice $2/3$. A partire dal punto x_2 sono stati necessari solo 6 iterazioni per raggiungere la precisione di circa $1E-15$. Chiaramente il metodo è risultato estremamente veloce. Una stima della velocità media possiamo dedurla per mezzo delle note formule.

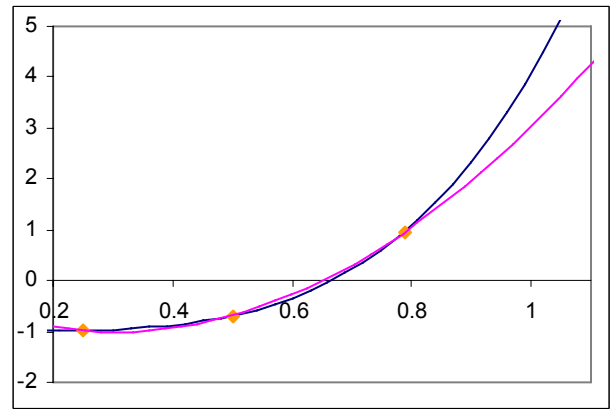
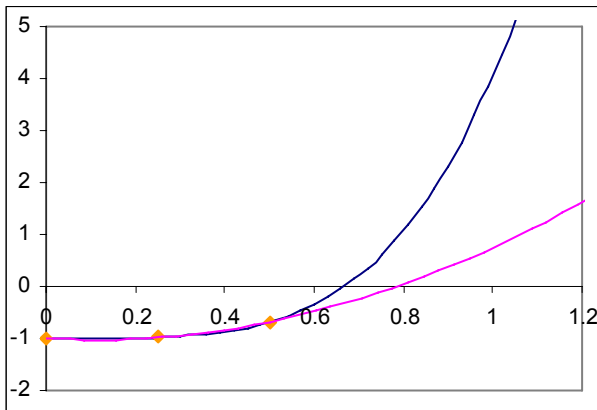
test $f(x) = (3x/2)^4 - 1$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 6 iterazioni
 L'ordine di convergenza medio è di $p = 1.9$
 La costante asintotica misurata è $c \cong 1.4$
 Costo tot = $6 \cdot 27 = 135$



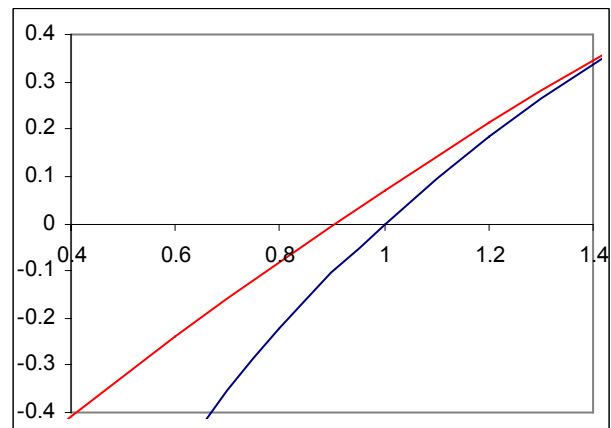
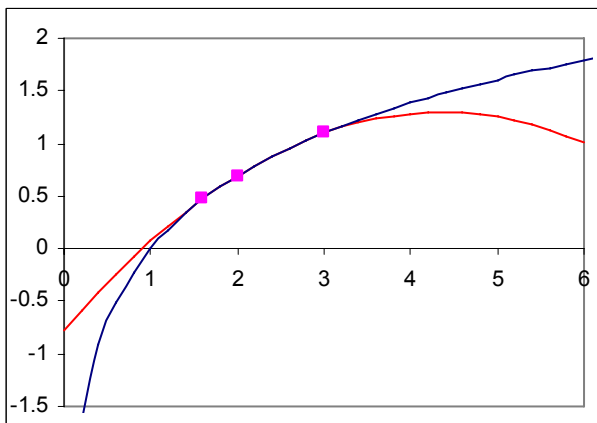
Non è necessario che la radice si contenuta nell'intervallo di partenza. Ripetiamo la ricerca partendo con la terna $[0, 0.25, 0.5]$. Il metodo converge ugualmente in circa 7 iterazioni. I grafici seguenti mostrano la prima e seconda parabola del processo iterativo

La prima è relativa alla terna iniziale $[0, 0.25, 0.5]$ che dà come intersezione $x_3 \cong 0.8$; la seconda è relativa alla terna $[0.25, 0.5, 0.8]$



Si vede che la prima iterazione dà un valore $x_3 \cong 0.8$ relativamente distante dalla radice ma, nonostante questo la terna, x_3 , insieme a x_2 e x_1 fornisce il successivo valore x_4 molto vicino alla vera radice. Il metodo sembra molto stabile e in grado di autocorreggersi contro eventuali errori introdotti nella successione.

Esempio. Approssimare lo zero ($x = 1$) della funzione $f(x) = \log(x)$ partendo dai punti $[3, 2, 1.6]$, usando l'interpolazione parabolica.



Le formule forniscono

a	b	c	Δ	x interp.
-0.1088	0.94973	-0.7709	0.5663	0.9057

Come si vede anche con una sola iterazione il valore approssimato è già buono, considerando che l'approssimazione rettilinea sui i punti più vicini avrebbe dato solamente $x_3 \cong 0.75$

Il metodo non conserva il bracketing e quindi può essere innescato con qualunque terna di punti. Tuttavia se l'intervallo iniziale contiene la radice, si è visto sperimentalmente che i successivi punti di approssimazione tendono a rimanere all'interno dell'intervallo iniziale (quasi-bracketing)

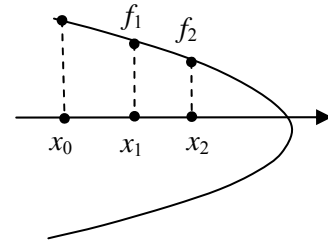
Di questo metodo non si conosce nessuna prova di garanzia di convergenza. Tuttavia dalle prove sperimentali si è visto che il metodo raramente fallisce. Quindi può essere annoverato senz'altro fra i metodi robusti. Si è misurato anche un'alta velocità di convergenza. Quindi il metodo risulta - caso raro - contemporaneamente veloce e robusto.

Forse l'unico difetto di questo metodo è la sua complessità.

Metodo della Parabola Inversa

Questo metodo è concettualmente analogo al caso precedente. Si differenzia per usare una parabola ad asse orizzontale, o inversa, anziché quella ad asse verticale

Dati tre punti (x_0, f_0) , (x_1, f_1) , (x_2, f_2) determiniamo l'intersezione con l'asse x della parabola ad asse orizzontale che interpola i punti dati. Osserviamo che in questo caso l'intersezione con l'asse x esiste sempre, a differenza della parabola verticale.



Sotto queste ipotesi, la parabola generica sarà $x = a y^2 + b y + c$. I coefficienti vengono determinati imponendo i vincoli sui punti $x(y_0) = x_0$, $x(y_1) = x_1$, $x(y_2) = x_2$

I coefficienti della parabola possono essere ottenuti con le formule del capitolo precedente dopo aver invertito le coordinate x e y. Tuttavia poiché siamo interessati solo all'intersezione con l'asse x, osserviamo che essa è data direttamente dal coefficiente "c" e quindi è conveniente ricavare direttamente solo tale coefficiente.

Impostiamo il sistema

$$\begin{cases} x_0 = a y_0^2 + b y_0 + c \\ x_1 = a y_1^2 + b y_1 + c \\ x_2 = a y_2^2 + b y_2 + c \end{cases}$$

Moltiplichiamo la prima per y_1 e la seconda per y_0 e sottraiamo. Moltiplichiamo la seconda per y_3 e la terza per y_1 e sottraiamo. Abbiamo il sistema

$$\begin{cases} x_0 y_1 - x_1 y_0 = y_0 y_1 (y_0 - y_1) a - (y_0 - y_1) c \\ x_1 y_2 - x_2 y_1 = y_1 y_2 (y_1 - y_2) a - (y_1 - y_2) c \end{cases}$$

Ovvero

$$\begin{cases} y_0 y_1 a - c = (x_0 y_1 - x_1 y_0) / (y_0 - y_1) \\ y_1 y_2 a - c = (x_1 y_2 - x_2 y_1) / (y_1 - y_2) \end{cases}$$

Moltiplicando la prima per y_2 e la seconda per y_0 e sottraendo si ha

$$c = \frac{1}{y_2 - y_0} \left[y_0 \left(\frac{x_1 y_2 - x_2 y_1}{y_1 - y_2} \right) - y_2 \left(\frac{x_0 y_1 - x_1 y_0}{y_0 - y_1} \right) \right] \quad (1)$$

Costo computazionale: La formula (1) contiene 15 operazioni e permette di calcolare direttamente i valori della successione iterativa; quindi il costo totale del metodo è $15 \text{ op} + 1 \text{ f}$

Il costo sensibilmente inferiore è il vantaggio principale del metodo della parabola inversa rispetto alla parabola verticale.

Applichiamo questo metodo al caso di test $f(x) = (3x/2)^3 - 1$, partendo dalla terna $[0, 1, 0.5]$

x	y
0	-1
1	2.375
0.5	-0.57813

Con il metodo dello scambio delle coordinate si ricava la parabola inversa

⇒

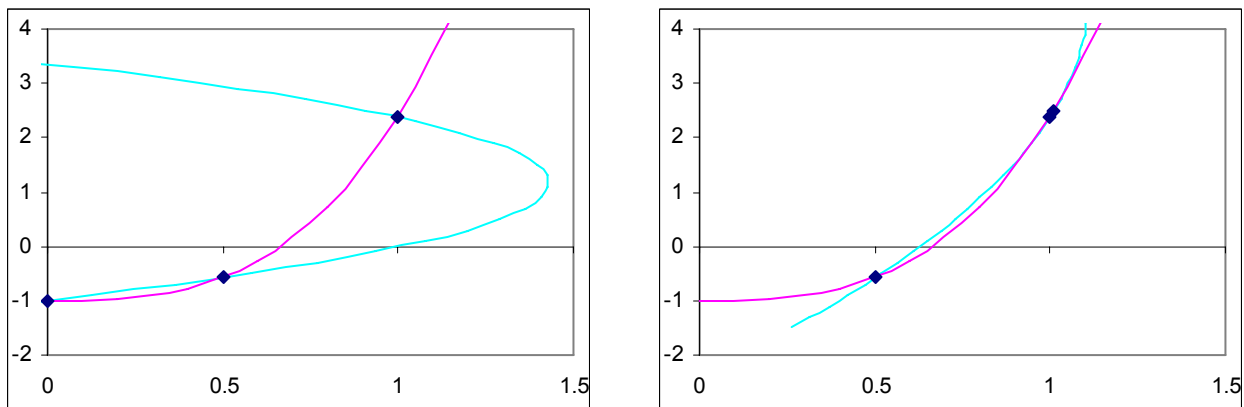
x	y
-1	0
2.375	1
-0.57813	0.5

La parabola inversa interpolante ha i coefficienti

a	b	c
-0.301	0.710	1.011

Quindi la prima approssimazione del valore della radice è $x_3 \cong 1.011$. L'errore è considerevole e sembra che la successione tenda ad allontanarsi dalla radice. Tuttavia la seconda iterata effettuata con con la terna $[1, 0.5, 1.11]$ dà come valore $x_3 \cong 0.63$, assai vicino alla vera radice. Questo comportamento è tipico del metodo delle parabole. Anche i presenza di forti errori la successione sembra autocorreggersi. L'intrinseca stabilità, unito alla elevata velocità di convergenza, è un dei punti di maggior forza del metodo delle parabole.

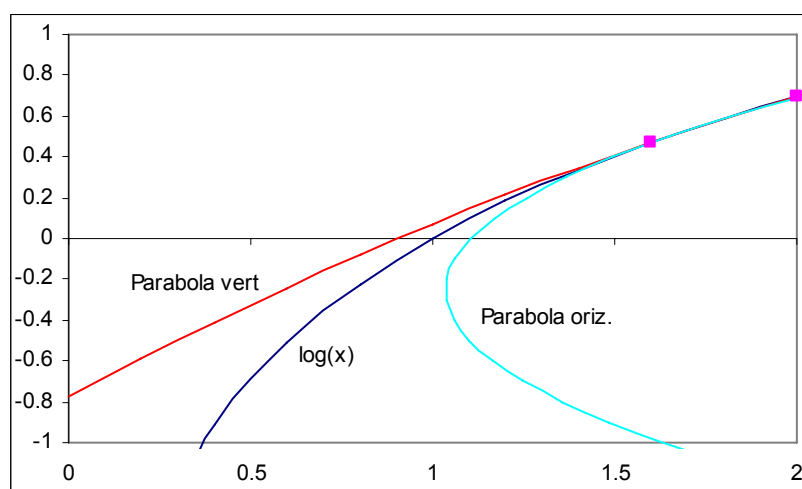
Il grafici seguenti mostrano le prime due parabole della sequenza



Esempio. Approssimare lo zero ($x = 1$) della funzione $f(x) = \log(x)$ partendo dai punti $[3, 2, 1.6]$, usando l'interpolazione parabolica inversa. Le formule forniscono

a	b	c	x interp.
1.071	0.545	1.1066	1.1066

Confrontiamo l'interpolazione con quella del metodo della parabola ad asse verticale



Come si vede, dal punto di vista della precisione, i valori interpolati della parabola verticale e della parabola orizzontale si equivalgono ($\pm 10\%$).

Metodo della Frazione Lineare

L'interpolazione per mezzo della frazione lineare è particolarmente utile per la ricerca degli zeri in prossimità degli asintoti verticali

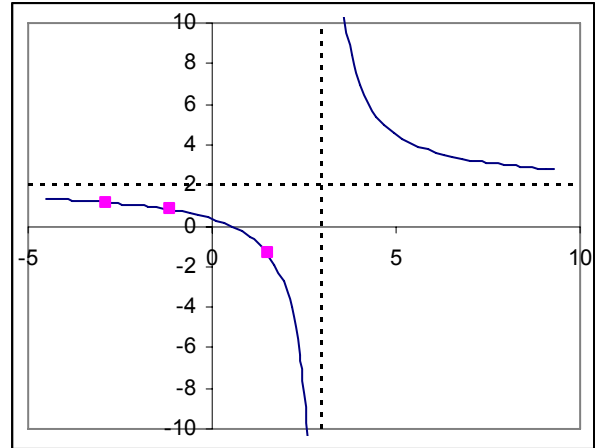
Dati tre punti $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ vogliamo determinare la frazione lineare del tipo

$$y = \frac{b}{x - a} + c \quad (1)$$

Che passa per i tre punti dati

Si tratta di un problema d'interpolazione con frazione lineare che è risolto quando si conoscono i parametri incogniti a, b, c

Tali parametri possono essere trovati imponendo i vincoli sui tre punti e risolvendo il sistema nelle incognite a, b, c



Si osserva che il parametro "a" rappresenta il punto per cui passa l'asintoto verticale; il parametro "c" rappresenta il punto per cui passa l'asintoto orizzontale; il parametro "b" è un coefficiente di dilatazione verticale.

I valori degli asintoti "a" e "c" possono essere trovati per mezzo del seguente sistema

$$\begin{bmatrix} y_1 - y_0 & x_1 - x_0 \\ y_2 - y_1 & x_2 - x_1 \end{bmatrix} \cdot \begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} x_1 y_1 - x_0 y_0 \\ x_2 y_2 - x_1 y_1 \end{bmatrix}$$

Che risolto dà

$$a = -\frac{x_0 x_1 (y_0 - y_1) + x_0 x_2 (y_2 - y_0) + x_1 x_2 (y_1 - y_2)}{d}$$

$$c = \frac{x_0 y_0 (y_1 - y_2) + x_1 y_1 (y_2 - y_0) + x_2 y_2 (y_0 - y_1)}{d}$$

dove

$$d = x_0 (y_1 - y_2) + x_1 (y_2 - y_0) + x_2 (y_0 - y_1)$$

Il coefficiente b viene ricavato poi per mezzo della relazione

$$b = (x_1 - a)(y_1 - c)$$

Le relazioni precedenti permettono di ricavare tutti i parametri incogniti del modello razionale (1) e quindi di tracciarne la funzione. Tuttavia per conoscere solo lo zero della funzione, che esiste sempre se $c \neq 0$, è più efficiente usare la seguente formula

$$\tilde{x} = x_2 + \frac{(y_0 - y_1)y_2}{\left(\frac{y_0 - y_2}{x_0 - x_2}\right)y_1 - \left(\frac{y_1 - y_2}{x_1 - x_2}\right)y_0} \quad (2)$$

Per mezzo di questa formula è possibile creare un metodo iterativo per approssimare la radici di una qualunque funzione. Infatti posto $x_3 = \tilde{x}$ e $y_3 = f(x_3)$ si ripete il calcolo con la terna $[x_1, x_2, x_3]$

Costo computazionale: la formula contiene 13 op. Quindi il costo per iterazione sarà $13 \text{ op} + 1 \text{ f}$

Ad esempio trovare la frazione lineare interpolante i seguenti tre punti

x	y
-2.9	1.152542
-1.14	0.792271
1.5	-1.33333

I punti sono rappresentati nel grafico precedente.

Applicando le formule ricavate si trova

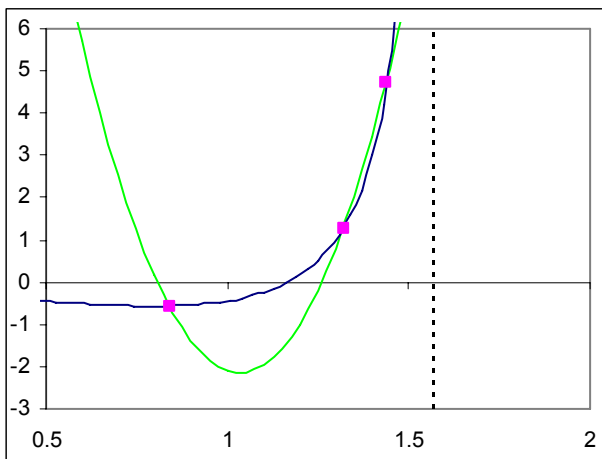
$$a = 3, c = 2, b = 5, \tilde{x} = 0.5$$

La formula d'interpolazione fratta viene usata soprattutto nelle vicinanze dei poli.

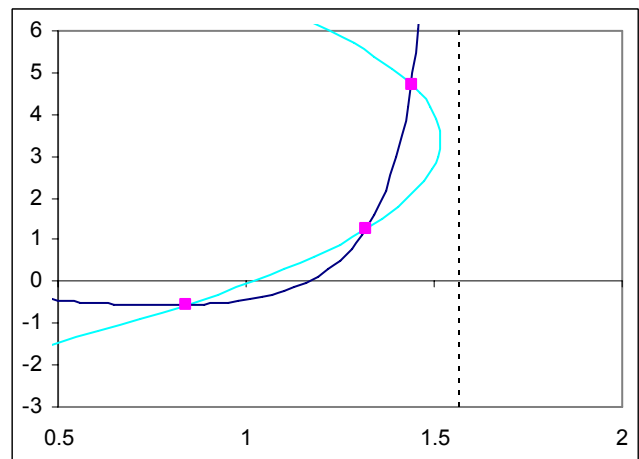
Ad esempio si voglia approssimare lo zero della funzione $f(x) = \tan(x) - 2x$ nell'intervallo $[0, 1.5]$. Si noti che la funzione ha un polo in prossimità di $x \cong 1.57$. Usiamo l'interpolazione parabolica verticale e orizzontale con i punti a fianco indicati

x	y
0.84	-0.564368
1.32	1.2633478
1.44	4.7218261

I risultati sono mostrati nei seguenti grafici. Entrambe le approssimazioni sono piuttosto scadenti. Il motivo è nella vicinanza del polo; i polinomi come le parabole non avendo poli non possono "seguire" la funzione in modo sufficientemente accurato. Il "distacco" appare infatti evidente in entrambi grafici

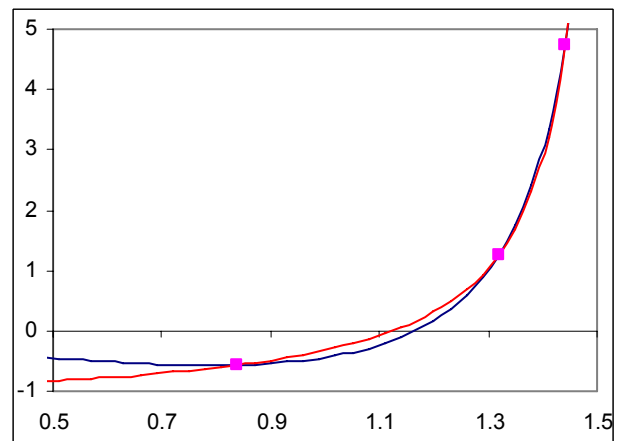
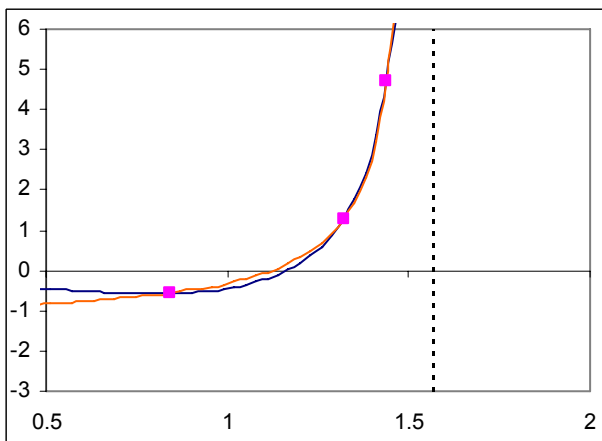


Interpolazione con parabola verticale



Interpolazione con parabola orizzontale

L'interpolazione fratta invece, in questa situazione, dà un'approssimazione molto più accurata.



La miglior approssimazione appare conseguente in quanto l'interpolazione razionale è stata creata per questo scopo. Tuttavia è sorprendente che l'interpolazione fratta funzioni ugualmente bene anche in altre situazioni più comuni tanto da essere un metodo di ricerca degli zeri competitivo con altri. Vediamo infatti come lavora con la funzione test

test $f(x) = (3x/2)^3 - 1$

Punto d'innescò $x_0 = 1$

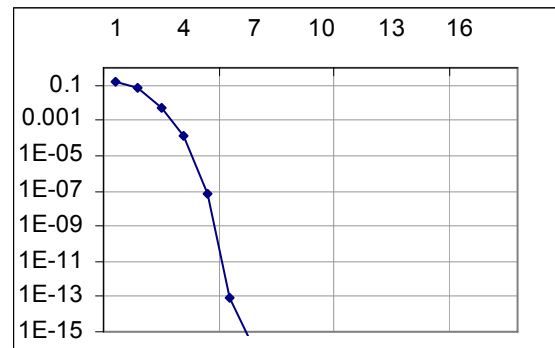
Il metodo converge alla radice $x = 2/3$ con un

errore di circa $e \cong 1E-15$ in 6 iterazioni

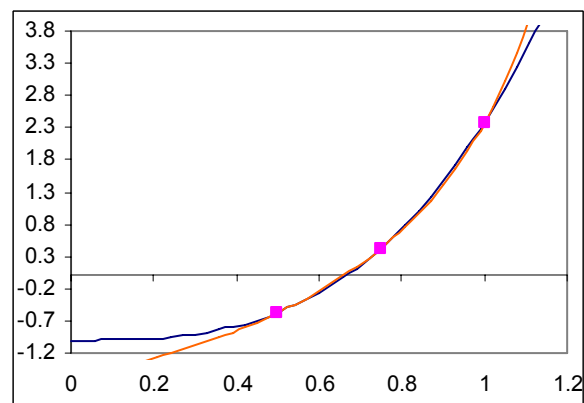
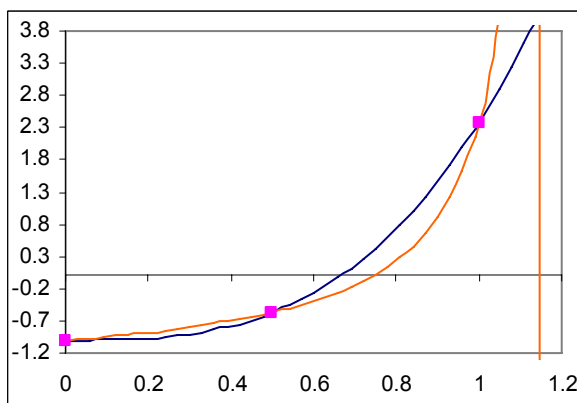
L'ordine di convergenza medio è di $p = 1.9$

La costante asintotica misurata è $c \cong 1.5$

Costo tot = $6 \cdot 18 = 108$



Nei grafici seguenti sono visualizzate le prime due iterazioni: la prima interpolazione fratta è stata ottenuta con la terna $[0, 0.5, 1]$ che dà il valore approssimato $x_3 \cong 0.75$; la seconda interpolazione è stata ottenuta con la terna $[0.5, 1, 0.75]$ che dà il valore $x_4 \cong 0.661$ con un errore di circa $5E-3$.



Come si vede la convergenza del metodo rimane molto alta e paragonabile a quella delle parabole anche se con un costo sensibilmente minore. Per contro, le prove sperimentali hanno rilevato una minor robustezza

Questo metodo va in crisi, come molti altri, con le radici multiple.

Metodo Muller

E' un altro antico metodo che sfrutta l'interpolazione parabolica su tre punti.

Le formule originali sono

$$q = (x_i - x_{i-1}) / (x_{i-1} - x_{i-2})$$

$$a = q f_i - q(1 + q) f_{i-1} + q^2 f_{i-2}$$

$$b = (1 + 2q) f_i - (1 + q)^2 f_{i-1} + q^2 f_{i-2}$$

$$c = (1 + q) f_i$$

$$\Delta = b^2 - 4ac \quad , \quad \text{se } \Delta \leq 0 \text{ allora si pone } \Delta = 0$$

$$x_{i+1} = x_i + (x_i - x_{i-1}) \frac{2c}{b + \text{sgn}(b)\sqrt{\Delta}}$$

Costo computazionale: Il costo è uno dei più alti fra i metodi iterativi: 33 op + 1 f

La convergenza, la stabilità e il comportamento generale sono eccellenti come quelli della parabola verticale anche se il costo è più del 50% maggiore.

Metodo Star

Il metodo di Newton richiede la conoscenza della derivata nel punto d'interpolazione. Vari metodi sono stati ideati per avere una stima della derivata senza ricorrere al calcolo diretto né ad una sua approssimazione con le differenze finite. Se si usano due punti della sequenza (quello attuale più quello precedente) si ottiene come abbiamo visto il metodo della secante. Se si usano tre punti (quello attuale più i due precedenti) si ha il cosiddetto metodo "Star"¹ o anche delle tre secanti.

Assumiamo per semplicità tre punti x_0, x_1, x_2 equispaziati di passo h . In tal caso la derivata prima nel punto x_2 può essere approssimata dalla nota formula

$$f'(x_2) \cong \frac{f_0 - 4f_1 + 3f_2}{2h} \quad (1)$$

Ma la (1) può essere riscritta come

$$\begin{aligned} \frac{f_0 - 4f_1 + 3f_2}{2h} &= \frac{f_2 - f_0 + 2f_2 - 4f_1 + 2f_0}{2h} = \frac{f_2 - f_0}{2h} + \frac{2f_2 - 2f_1 - 2f_1 + 2f_0}{2h} = \\ &= \frac{f_2 - f_0}{2h} + \frac{(f_2 - f_1) - (f_1 - f_0)}{h} = d_{20} + d_{21} - d_{10} \end{aligned}$$

Quindi

$$f'(x_2) = d_{20} + d_{21} - d_{10} \quad (2)$$

dove d_{ij} è la pendenza della retta fra i punti i e j . Cioè:

$$d_{20} = \frac{f_2 - f_0}{x_2 - x_0} \quad d_{21} = \frac{f_2 - f_1}{x_2 - x_1} \quad d_{10} = \frac{f_1 - f_0}{x_1 - x_0}$$

Sostituendo (2) nella formula di Newton si ha

$$x_3 = x_2 + \frac{f_2}{f_2'} = x_2 + \frac{f_2}{d_{20} + d_{21} - d_{10}} \quad (3)$$

Questa relazione è la formula iterative del metodo Star. Nei casi pratici i punti non sono uniformemente spazati e quindi la (2) darà un valore tanto più approssimato tanto più ci allontaniamo da questa condizione. Comunque si tratta di una approssimazione nell'approssimazione e abbiamo mostrato che la precisione della derivata non influenza la precisione del metodo di Newton ma solo la sua velocità di convergenza.

Costo computazionale. Il calcolo di d_{10} non viene conteggiato perchè ad ogni iterazione il suo valore è quello di d_{21} della iterata precedente. Quindi il costo del metodo è $10 + 1 f$

Questo metodo è stato derivato in modo euristico come approssimazione della formula di Newton. Ci aspetteremo quindi delle caratteristiche considerevoli ma certamente inferiori a questo. Ebbene, per un caso raro, il metodo "Star" ha presentato alle prove sperimentali delle caratteristiche globali medie di velocità, precisione, e robustezza persino superiori a quello di Newton. Vediamo come lavora in un caso di test.

¹ Il metodo è stato codificato dal Traub come "Star E21". ("Iterative Methods for the solution of Equations", J. F. Traub, Prentice Hill, 1964)

test $f(x) = (3x/2)^3 - 1$

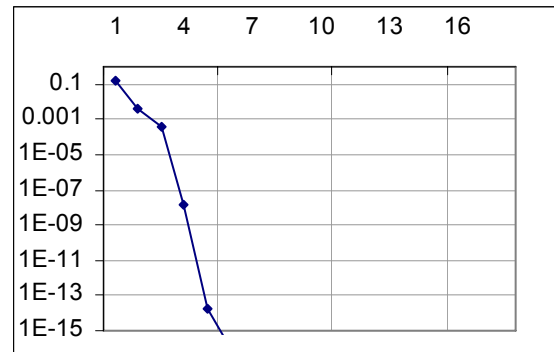
Terna d'innesco $[0, 1, 0.5]$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 5 iterazioni

L'ordine di convergenza medio è di $p = 1.6$

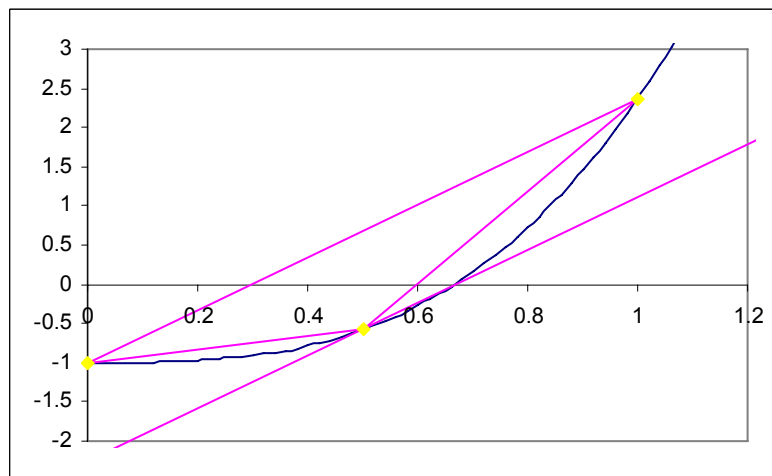
La costante asintotica misurata è $c \cong 0.13$

Costo tot = $5 \cdot 15 = 75$



Come si vede la convergenza è "bruciante". Il metodo è riuscito a sorpassare in velocità, se pur di poco, persino il metodo di Newton del 2° ordine. Solo il metodo di Halley del 3° ordine è riuscito a fare meglio (4 iterazioni). Questo sorprendente risultato è in parte spiegabile con la bassa costante di convergenza $c = 0.13$, che unito all'ordine di convergenza medio di circa $p = 1.6$ ha prodotto questo brillante risultato

Nel grafico è illustrato il primo passo d'iterazione con le tre secanti e la retta interpolante passante per il punto $(0.5, -0.578)$.



Il punto d'interpolazione, come si vede anche dal grafico, è molto accurato già dalla prima iterazione ($x_3 \cong 0.6712$ err < 0.7%). Il metodo di Newton calcolato in 0.5 avrebbe dato un errore 10 volte più elevato (0.73 err < 9%). Chiaramente un metodo che esibisce simili performance merita una attenta sperimentazione.

test $f(x) = (3x/2)^6 - 1$

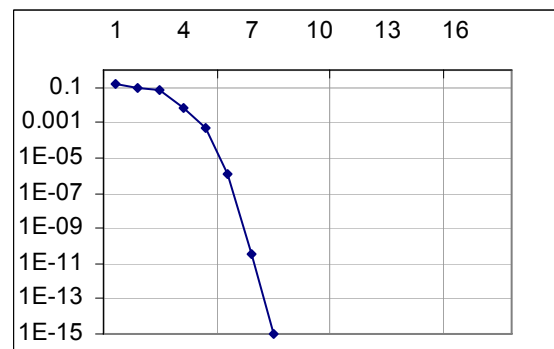
Punti d'innesco $[0, 1, 0.5]$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 7 iterazioni

L'ordine di convergenza medio è di $p = 1.5$

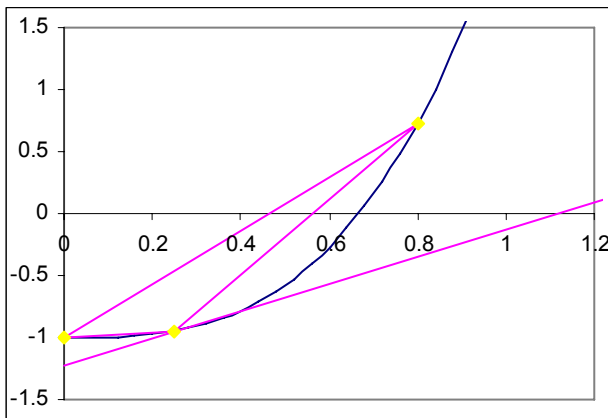
La costante asintotica misurata è $c \cong 0.4$

Costo tot = $7 \cdot 15 = 105$

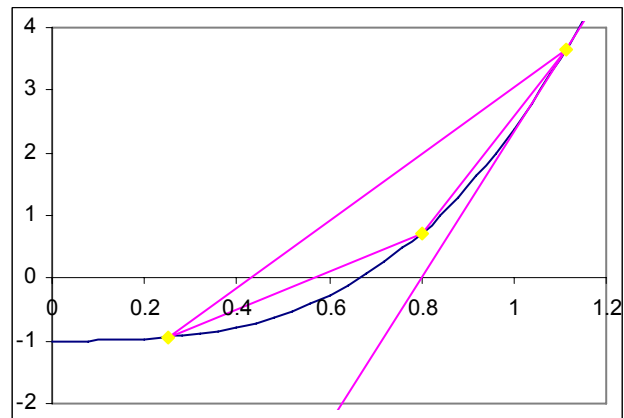


Anche di questo algoritmo non si ha garanzia né della convergenza né del bracketing. Tuttavia è molto difficile che l'algoritmo fallisca o esca dall'intervallo iniziale. Nei rari casi in cui questo avviene l'algoritmo tende ad autocorreggersi similmente a quanto avviene nel metodo della parabola.

Ripetiamo ad esempio il test per la funzione test $f(x) = (3x/2)^3 - 1$ partendo da $[0, 0.8, 0.25]$. L'intervallo di partenza è 0.8 che contiene la radice $2/3$; alla prima iterazione si ottiene il valore $x_3 \cong 0.4$, fuori dall'intervallo. Alla seconda iterazione si ottiene però $x_3 \cong 0.8$ che indica la tendenza dell'algoritmo ad autostabilizzarsi per tornare poi a convergere alla radice.



1° iterazione



2° iterazione

Dalle prove effettuate è emerso un algoritmo estremamente robusto, veloce, efficiente e molto stabile in svariate situazioni. Tenuto conto del suo costo modesto e del fatto che non necessita delle derivate, è un risultato senz'altro straordinario.

Metodo Chebyshev

Si tratta di un metodo del 3 ordine ad alta velocità di convergenza, conosciuto come il metodo di Householder o formula di Chebyshev. Usa sia la derivata prima che la derivata seconda. Può essere ricavato in vari modi. La più immediata è l'espansione polinomiale di Taylor di 2° grado

$$f(x) \cong f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2} f''(x_i)(x - x_i)^2$$

L'intersezione x_{i+1} con l'asse x si trova imponendo $f(x_{i+1}) = 0$

$$\frac{1}{2} f'' h^2 + f' h + f = 0$$

dove $h = x_{i+1} - x_i$, $f'' = f''(x_i)$, $f' = f'(x_i)$, $f = f(x_i)$. Ammettiamo $f'' > 0$ e risolviamo l'equazione di secondo grado in h

$$\begin{aligned} h &= \frac{-f' \pm \sqrt{(f')^2 - 2f f''}}{f''} = -\frac{f'}{f''} \pm \sqrt{\frac{(f')^2 - 2f f''}{(f'')^2}} = -\frac{f'}{f''} \pm \sqrt{\left(\frac{f'}{f''}\right)^2 - 2\frac{f}{f''}} \\ &= -\frac{f'}{f''} \pm \sqrt{\left(\frac{f'}{f''}\right)^2 \left(1 - \frac{2ff''}{(f')^2}\right)} = -\frac{f'}{f''} \pm \left|\frac{f'}{f''}\right| \sqrt{1 - \alpha} \quad , \quad \text{con } \alpha = \frac{2ff''}{(f')^2} \end{aligned}$$

Ricordando lo sviluppo in serie della radice $\sqrt{1 - \alpha} \cong 1 - \frac{1}{2}\alpha - \frac{1}{8}\alpha^2$

$$h = -\frac{f'}{f''} \pm \left|\frac{f'}{f''}\right| \left(1 - \frac{1}{2}\alpha - \frac{1}{8}\alpha^2\right) \quad (1)$$

Scegliamo il segno per rendere h minimo; cioè scegliamo $+$ se $f' > 0$, $-$ se $f' < 0$

1) caso $f' > 0$

$$h = -\frac{f'}{f''} + \frac{f'}{f''} \left(1 - \frac{1}{2}\alpha - \frac{1}{8}\alpha^2\right) = -\frac{f'}{f''} \cdot \frac{\alpha}{2} \left(1 + \frac{\alpha}{4}\right) \quad (2)$$

Sostituendo α e semplificando si ha

$$h = -\frac{f'}{f''} \cdot \frac{1}{2} \cdot \frac{2ff''}{(f')^2} \left(1 + \frac{1}{4} \cdot \frac{2ff''}{(f')^2}\right) = -\frac{f}{f'} \left(1 + \frac{f f''}{2(f')^2}\right) \quad (3)$$

2) caso $f' < 0$

$$h = -\frac{f'}{f''} - \left|\frac{f'}{f''}\right| \left(1 - \frac{1}{2}\alpha - \frac{1}{8}\alpha^2\right)$$

Essendo $\left|\frac{f'}{f''}\right| = -\frac{f'}{f''} \Rightarrow h = -\frac{f'}{f''} + \frac{f'}{f''} \left(1 - \frac{1}{2}\alpha - \frac{1}{8}\alpha^2\right) = -\frac{f'}{f''} \cdot \frac{\alpha}{2} \left(1 + \frac{\alpha}{4}\right) \quad (4)$

Ma la 4) e la 2) sono uguali; quindi anche in questo caso la formula dell'incremento h è identica alla 3) e cioè:

$$x_{i+1} - x_i = -\frac{f}{f'} \left(1 + \frac{f f''}{2(f')^2}\right) \quad (5)$$

Si osservi che questa formula è diversa, se pur molto simile, da quella di Halley

Costo computazionale: Il costo del metodo di Chebyshev è determinato in massima parte dal calcolo della derivate prima e seconda. Per funzioni comuni possiamo stimare il costo equivalente a quello della funzione stessa, per cui il costo per iterazione è medio-alto: 7 op è 3 f
Nel caso che di approssimazione mediante le differenze divise il costo sale a: 7 op + 5 f

Metodo Chebyshev con differenze finite (Secantx)

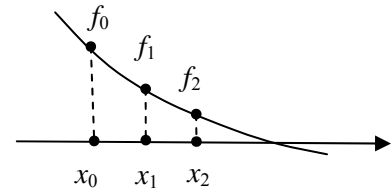
Anche per la formula del 3 ordine di Chebyshev-Householder, così come per quella di Halley, è stato ideato un ingegnoso metodo per evitare il calcolo delle derivate. Il metodo viene chiamato in modi diversi¹ ma sostanzialmente è il metodo di Chebyshev in cui le derivate sono sostituite con le differenze finite calcolate nei punti della successione stessa.

Dati tre punti d'innescio $(x_0, f_0), (x_1, f_1), (x_2, f_2)$

Posto

$$d_1 = (f_1 - f_0)/(x_1 - x_0)$$

$$d_2 = (f_2 - f_1)/(x_2 - x_1)$$



La formula iterativa è

$$x_3 - x_2 = -\frac{f_2}{d_2} - \frac{f_1 f_2}{f_2 - f_0} \left(\frac{1}{d_2} - \frac{1}{d_1} \right) \tag{1}$$

Possiamo far vedere che se i punti della successione x_0, x_1, x_2 sono molto vicini ed equispaziati, cioè tale che $\Delta x \cong x_1 - x_0 \cong x_2 - x_1$, allora la (1) approssima la formula di Chebyshev

Infatti se i punti sono molto vicini allora $d_1 \cong f_1'$ e $d_2 \cong f_2'$

Inoltre $\frac{f_2}{d_2} \cong \frac{f_2}{f_2'} \cong \frac{f}{f'}$, $f_1 f_2 \cong f^2$, $f_2 - f_0 \cong f_0'(x_2 - x_0) \cong 2f'\Delta x$

$$\frac{1}{d_2} - \frac{1}{d_1} \cong \frac{1}{f_2'} - \frac{1}{f_1'} = -\frac{f_2' - f_1'}{f_1' f_2'}$$

Approssimando $f_1' f_2' \cong (f')^2$, $f_2' - f_1' \cong f_1''(x_2 - x_1) \cong f''\Delta x$

E sostituendo tutte queste approssimazioni in 1) si ha

$$x_3 - x_2 = -\frac{f}{f'} + \frac{f^2}{2f'\Delta x} \cdot \left(-\frac{f_2' - f_1'}{f_1' f_2'} \right) = -\frac{f}{f'} + \frac{f^2}{2f'\Delta x} \cdot \left(-\frac{f''\Delta x}{(f')^2} \right) = -\frac{f}{f'} - \frac{f^2 f''}{2(f')^3} = -\frac{f}{f'} \left(1 + \frac{f f''}{2(f')^2} \right)$$

Che è appunto la formula Chebyshev

Costo computazionale. Il calcolo di d_1 non viene conteggiato perchè ad ogni iterazione il suo valore è quello di d_2 della iterata precedente. Quindi il costo del metodo è $13 + 1 f$

test $f(x) = (3x/2)^3 - 1$

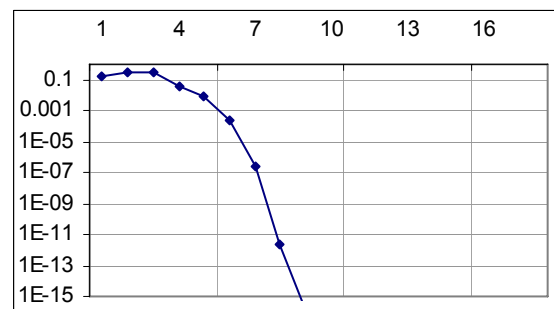
Terna d'innescio $[0, 1, 0.5]$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1E-15$ in 8 iterazioni

L'ordine di convergenza medio è di $p = 1.8$

La costante asintotica misurata è $c \cong 1.9$

Costo tot = $8 \cdot 18 = 144$



¹ Il Traub in "Iterative Methods for the solution of Equations" chiama questo metodo come "Secantx", ovvero metodo della secante estesa.

Metodo Wijngaardern- Dekker-Brent

Questo eccellente metodo combina la robustezza dell' algoritmo di bisezione con la velocità del metodo della secante e dell' interpolazione parabolica inversa. Sviluppato al Centro Matematico di Amsterdam da Wijngaardern, Dekker ed altri, e migliorato, più tardi, da Brent che ne ha garantito anche la convergenza sotto le stesse condizioni del metodo di bisezione.

Si tratta di un metodo cosiddetto "ibrido", cioè che usa una combinazione di tre differenti metodi: bisezione, secante, parabola inversa. Si tratta di un metodo abbastanza complicato ed è difficile spiegare in poche righe l' intero processo.

Le formule usate dall' algoritmo sono la formula di bisezione, l' interpolazione lineare per due punti e l' interpolazione di Lagrange di 2° grado per tre punti

$$x_3 = \frac{f_1 f_2}{(f_0 - f_1)(f_0 - f_1)} x_0 + \frac{f_0 f_2}{(f_1 - f_0)(f_1 - f_2)} x_1 + \frac{f_1 f_0}{(f_2 - f_1)(f_2 - f_0)} x_2$$

Poiché l' algoritmo necessita ad ogni iterazione di controllare certe variabili intermedie per decidere quando usare un metodo anziché un altro, le formule vengono tradizionalmente riscritte in un altro modo più efficiente.

Posto: $R = f_1 / f_2$, $S = f_1 / f_0$, $T = f_0 / f_2$

Si calcolano i coefficienti P e Q

Se i segni degli ultimi due valori della successione sono discordi, cioè $f_2 f_1 < 0$, si effettua l' interpolazione lineare della secante altrimenti si effettua l' interpolazione parabolica inversa.

Cioè

se $f_2 f_1 < 0$

$$P = S \cdot (x_2 - x_1)$$

$$Q = 1 - S$$

altrimenti

$$P = S \cdot [T(R - T)(x_2 - x_1) - (1 - R)(x_1 - x_0)]$$

$$Q = (R - 1)(S - 1)(T - 1)$$

A questo punto l' algoritmo effettua un controllo per vedere se l' interpolazione è accettabile.

Se $2P < 3(x_2 - x_1) \cdot Q - |\varepsilon \cdot Q|$ oppure $2P < |S \cdot Q|$ allora si accetta l' interpolazione

$x_3 = x_1 + P/Q$ altrimenti si applica il metodo di bisezione e si pone $x_3 = (x_2 + x_1)/2$

Costo computazionale. Il conteggio deve comprendere anche le operazioni necessarie ai test, mentre le formule di interpolazione lineare e quadratica che sono mutuamente esclusive vengono conteggiate con un fattore di riduzione del 50%. Quindi il costo è $25 \text{ op} + 1 f$

test $f(x) = (3x/2)^3 - 1$

Intervallo $[0, 1]$

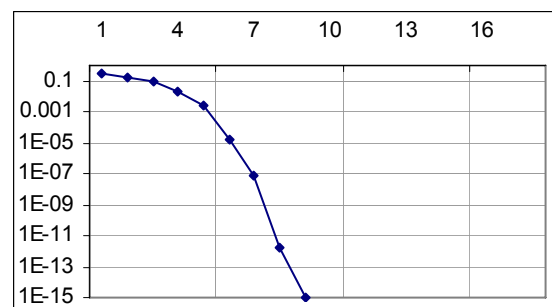
Il metodo converge alla radice $x = 2/3$ con un

errore di circa $\varepsilon \cong 1E-15$ in 9 iterazioni

L' ordine di convergenza medio è di $p = 1.5$

La costante asintotica misurata è $c \cong 1.3$

Costo tot = $9 \cdot 30 = 270$



test $f(x) = (3x/2)^6 - 1$

Intervallo $[0, 1]$

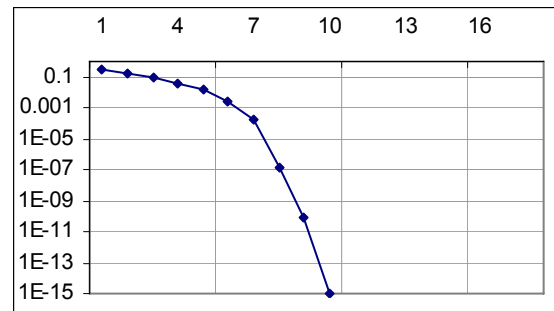
Il metodo converge alla radice $x = 2/3$ con un

errore di circa $e \cong 1E-15$ in 10 iterazioni

L'ordine di convergenza medio è di $p = 1.5$

La costante asintotica misurata è $c \cong 1.1$

Costo tot = $10 \cdot 30 = 300$



Questo test mostra il cambio di velocità del metodo evidenziato dalla differente pendenza della traiettoria. Nei primi passi viene usato il metodo di bisezione per avvicinarsi alla radice; poi l'algoritmo passa ai più veloci metodi della secante e parabola

Il costo è sensibilmente più alto degli altri metodi finora provati. Tuttavia si deve tener conto che questo metodo mantiene il bracketing e garantisce la convergenza.

Nei programmi rootfinder automatici questa qualità è di primaria importanza per cui i metodi "ibridi" come quello di Wijngaardern- Dekker-Brent sono molto usati. La perdita di efficienza viene compensata abbondantemente dalla robustezza e dalla stabilità. Inoltre questo metodo non ne necessita del calcolo delle derivate.

Metodo Rheinboldt (2)

Si tratta di un altro metodo "ibrido", cioè che usa una combinazione di tre differenti metodi: bisezione, secante, parabola inversa. Si tratta di un metodo abbastanza complicato, proposto da Rheinboldt¹. In questo documento è stato classificato come "Rheinboldt 2" per distinguerlo dai altri metodi proposti dallo stesso autore

Costo computazionale. il costo è di circa 28 op + 1 f

test $f(x) = (3x/2)^3 - 1$

Intervallo d'innescio $[0, 1]$

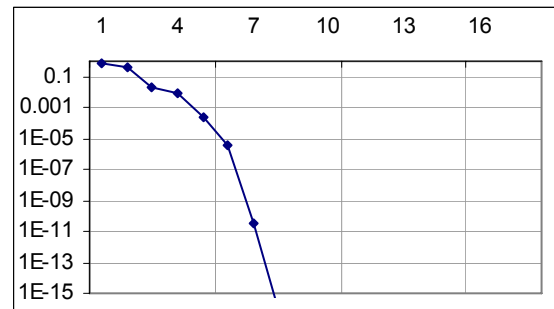
Il metodo converge alla radice $x = 2/3$ con un

errore di circa $e \cong 1E-15$ in 7 iterazioni

L'ordine di convergenza medio è di $p = 1.5$

La costante asintotica misurata è $c \cong 0.4$

Costo tot = $7 \cdot 33 = 231$



La costante asintotica relativamente bassa lo rende abbastanza veloce ed efficiente. Nelle prove sperimentali la stabilità è risultata inferiore al metodo di Brent.

¹ W C Rheinboldt, 'Algorithms for Finding Zeros of a Function,' UMAP Journal, 1981, pages 43 - 72.

Metodo Steffensen

Si tratta di un metodo senza derivate ad alta velocità di convergenza che viene innescato con un solo punto. Però richiede il calcolo di due volte la funzione per ogni iterazione

Le formule iterative sono:

$$g(x_n) = \frac{f(x_n + f(x_n)) - f(x_n)}{f(x_n)}$$

$$x_{n+1} = x_n + \frac{f(x_n)}{g(x_n)}$$

Costo computazionale. il costo è di circa $4 \text{ op} + 2 \text{ f}$

test $f(x) = (3x/2)^3 - 1$

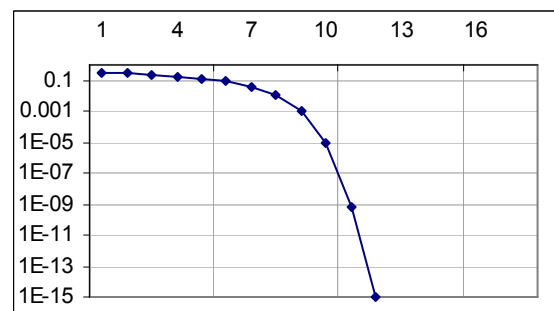
Punto d'innescio $x_0 = 1$

Il metodo converge alla radice $x = 2/3$ con un errore di circa $e \cong 1\text{E-}15$ in 11 iterazioni

L'ordine di convergenza medio è di $p = 1.7$

La costante asintotica misurata è $c \cong 2.2$

Costo tot = $11 * 14 = 154$



Si tratta di un metodo molto popolare e molto enfatizzato nei corsi teorici di calcolo numerico. Nonostante suo elevato ordine di convergenza, la costante asintotica abbastanza alta lo penalizza non poco, e quasi sempre viene superato in velocità da altri metodi. Nelle prove reali si è dimostrato anche oltremodo instabile. Da non scegliere per i processi automatici.

Formula Householder

Sotto opportune condizioni di regolarità delle funzione $f(x)$ e delle sue derivate, Householder trovò la seguente interessante formula iterativa generale¹

$$x_{i+1} = x_i + (p+1) \left(\frac{(1/f)^{(p)}}{(1/f)^{(p+1)}} \right)_{x=x_i} \quad (1)$$

dove

$$(1/f)^{(p)} = \frac{d^p}{dx^p} \left(\frac{1}{f(x)} \right) \quad \text{e} \quad (1/f)^{(0)} = 1/f$$

Questa formula ha ordine di convergenza $p+2$.

Per $p = 0$ si riottiene la formula di Newton. Infatti

$$(1/f)^{(0)} = 1/f \quad \text{e} \quad (1/f)^{(1)} = \frac{d}{dx} \left(\frac{1}{f(x)} \right) = -\frac{f'}{f^2}$$

Quindi sostituendo in (1)

$$\Delta x = \left(\frac{(1/f)^{(0)}}{(1/f)^{(1)}} \right) = -\left(\frac{1}{f} \frac{f^2}{f'} \right) = -\frac{f(x_i)}{f'(x_i)}$$

Per $p = 1$ si ritrova la formula di Halley. Infatti

$$(1/f)^{(2)} = \frac{d^2}{dx^2} \left(\frac{1}{f(x)} \right) = \frac{2(f')^2 - f \cdot f''}{f^3}$$

Quindi sostituendo in (1)

$$\Delta x = 2 \left(\frac{(1/f)^{(1)}}{(1/f)^{(2)}} \right) = -2 \left(\frac{f'}{f^2} \frac{f^3}{2(f')^2 - f \cdot f''} \right) = -\frac{f \cdot f'}{(f')^2 - \frac{1}{2} f \cdot f''}$$

Per $p = 2$ si ha la formula di Householder del quarto ordine

$$(1/f)^{(3)} = \frac{d^3}{dx^3} \left(\frac{1}{f(x)} \right) = \frac{6f f' f'' - f^2 f''' - 6(f')^3}{f^4}$$

Quindi sostituendo in (1)

$$\Delta x = 3 \left(\frac{(1/f)^{(2)}}{(1/f)^{(3)}} \right) = f \frac{(f')^2 - \frac{1}{2} f \cdot f''}{f f' f'' - \frac{1}{6} f^2 f''' - (f')^3}$$

¹ A. S. Householder, The Numerical Treatment of a Single Nonlinear Equation, McGraw-Hill, New-York, (1970)

Formula di Povposki

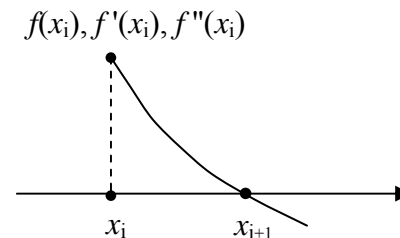
Per derivare metodi iterativi di ordine elevato per l'approssimazione numerica di radici ci sono vari modi. Un approccio originale e potente, originariamente sviluppato da D. B. Povposki, considera la seguente funzione parametrica.

$$y(x) = a + b(x - c)^e \tag{1}$$

dove [a, b, c, e] sono quattro parametri da determinare con le condizioni sui punti x_i e x_{i+1}

$$f(x_i) = y \quad , \quad f'(x_i) = y' \quad , \quad f''(x_i) = y''$$

$$f(x_{i+1}) = 0$$



Derivando si ha:

$$y'(x) = e \cdot b(x - c)^{e-1} \tag{2}$$

$$y''(x) = e(e - 1)b(x - c)^{e-2} \tag{3}$$

La condizione $y(x_{i+1}) = 0$ implica

$$y(x_{i+1}) = 0 \Rightarrow a + b(x_{i+1} - c)^e = 0 \Rightarrow x_{i+1} - c = \left(-\frac{a}{b}\right)^{1/e} \tag{4}$$

Cerchiamo di esprimere il rapporto a/b.

Dalla (1) si ricava $a = f - b(x - c)^e$

Dalla (2) si ricava $b = \frac{f'}{e(x - c)^{e-1}}$

Da cui

$$\begin{aligned} \left(-\frac{a}{b}\right)^{1/e} &= \left(-\frac{f - b(x - c)^e}{\frac{f'}{e(x - c)^{e-1}}}\right)^{1/e} = \left(\frac{eb(x - c)^e - ef}{(x - c)f'}(x - c)^e\right)^{1/e} = \\ &= (x - c) \left(\frac{eb(x - c)^{e-1}}{f'} - \frac{ef}{(x - c)f'}\right)^{1/e} \end{aligned}$$

Essendo $f' = eb(x - c)^{e-1}$, sostituendo si ha

$$\left(-\frac{a}{b}\right)^{1/e} = (x - c) \left(1 - \frac{ef}{(x - c)f'}\right)^{1/e} \tag{5}$$

Dividendo ora (2) con (3) si ha

$$\frac{y'}{y''} = \frac{f'}{f''} = \frac{x - c}{e - 1} \Rightarrow x - c = (e - 1) \frac{f'}{f''} \tag{6}$$

E sostituendo (6) in (5)

$$\left(-\frac{a}{b}\right)^{1/e} = (e - 1) \frac{f'}{f''} \left(1 - \frac{e}{e - 1} \frac{f f''}{(f')^2}\right)^{1/e} \tag{7}$$

Ora, sottraendo (4) e (6)

$$x_{i+1} - x_i = \left(-\frac{a}{b}\right)^{1/e} - (e-1) \frac{f'}{f''}$$

e tenendo conto di (7) si ottiene

$$x_{i+1} - x_i = (e-1) \frac{f'}{f''} \left[\left(1 - \frac{e}{e-1} \frac{f f''}{(f')^2}\right)^{1/e} - 1 \right]$$

Che è appunto la cosiddetta formula generale di Povposki

L'esponente "e" è il parametro più importante perché determina il tipo di metodo di approssimazione. Possiamo creare un metodo semplicemente variando il parametro "e"

Si possono riottenere anche le formule di Halley (interpolazione con parabola verticale) e Chebyshev (interpolazione con parabola orizzontale) dando rispettivamente i valori $e = -1$, $e = 1/2$

Infatti

per $e = -1$
$$x_{i+1} - x_i = \frac{f f''}{0.5 f f'' - (f')^2} \quad \text{Halley}$$

per $e = 1/2$
$$x_{i+1} - x_i = -\frac{f}{f'} \left(\frac{1 + 0.5 f f''}{(f')^2} \right) \quad \text{Chebyshev}$$

Se pur con passaggi un po' più complicati si può riottenere anche la formula di Newton

Infatti la formula si può scrivere come

$$(e-1) \frac{f'}{f''} \left[\left(1 - \frac{e}{e-1} \frac{f f''}{(f')^2}\right)^{1/e} - 1 \right] = \frac{f'}{f''} \left[(e-1)^{e-1} \left(e - 1 - e \frac{f f''}{(f')^2} \right)^{1/e} - (e-1) \right]$$

da cui passando al limite $e \rightarrow 1$ e ricordando che $\lim_{t \rightarrow 0} t^t = 1$ si ha

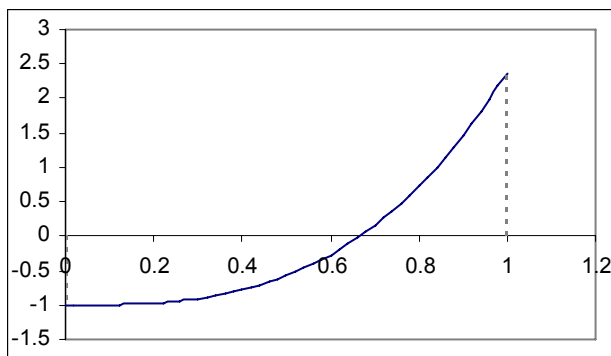
per $e \rightarrow 1$
$$x_{i+1} - x_i = -\frac{f'}{f''} \quad \text{Newton}$$

Test

I metodi rootfinding esposti nei precedenti paragrafi sono stati sottoposti a test per saggiare le loro caratteristiche di velocità, efficienza, e affidabilità. I vincoli imposti sono stati gli stessi per tutti i metodi: numero massimo d'iterazioni pari a 400 e errore assoluto della funzione $|f(x)| < 1E-15$. Se un metodo non convergeva nel numero assegnato di passi ovvero divergeva o convergeva ad una radice errata allora il test veniva considerato non superato. Nei casi di convergenza raggiunta, sono stati rilevati il numero di iterazioni, e il costo totale, per stilare le classifiche separate di velocità ed efficienza. La classifica dell'affidabilità è stata stilata in base al rapporto fra i successi e le prove totali.

Casi di Test

I casi di test sono stati scelti fra la tipologia di funzioni che s'incontrano comunemente nell'attività scientifica e sperimentale, evitando di proposito i casi patologici. Gli intervalli di partenza sono scelti volutamente molto ampi per saggiare meglio la robustezza dei metodi.



Test 01

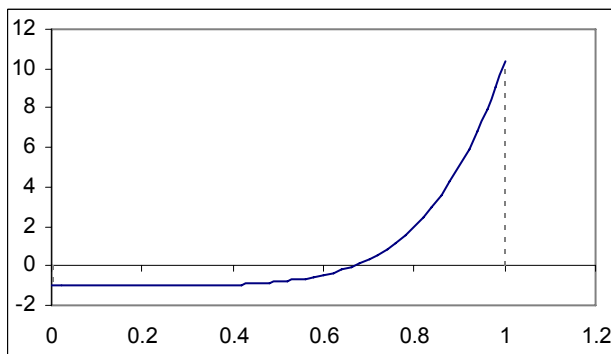
$$f(x) = (3x/2)^3 - 1$$

Presenta una radice $x = 2/3 \cong 0.666..$

Intervallo iniziale $[0, 1]$

Punto d'innescio $x_0 = 1$

La funzione è crescente con curvatura di segno costante in tutto l'intervallo.



Test 02

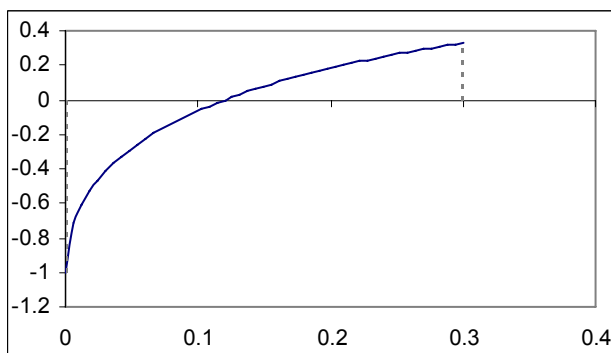
$$f(x) = (3x/2)^6 - 1$$

Presenta una radice $x = 2/3 \cong 0.666..$

Intervallo iniziale $[0, 1]$

Punto d'innescio $x_0 = 1$

La funzione è crescente con curvatura di segno costante e presenta tratti quasi costanti. Difficile per metodi come la "regula falsi" e la secante.



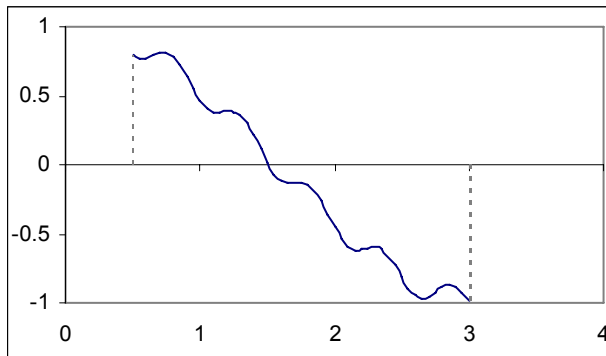
Test 03

$$f(x) = 1 - e^{-2\sqrt{x}}$$

Presenta una radice $x \cong 0.120111$

Intervallo iniziale $[0, 0.3]$

Punto d'innescio $x_0 = 0.3$



Test 04

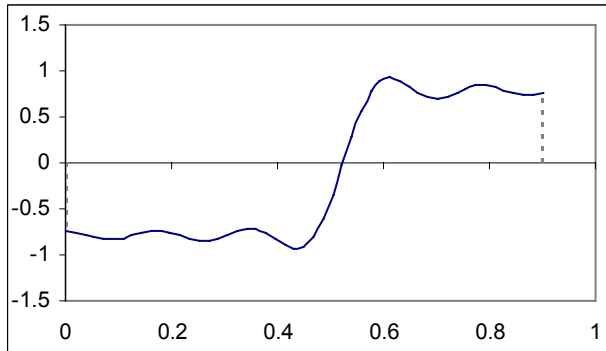
$$f(x) = \cos(x) - \frac{9}{100} \cos(12x)$$

Presenta una radice $x \approx 1.506350219$

Intervallo iniziale $[0.5, 3]$

Punto d'innescio $x_0 = 3$

Problemi con i metodi che usano le derivate



Test 05

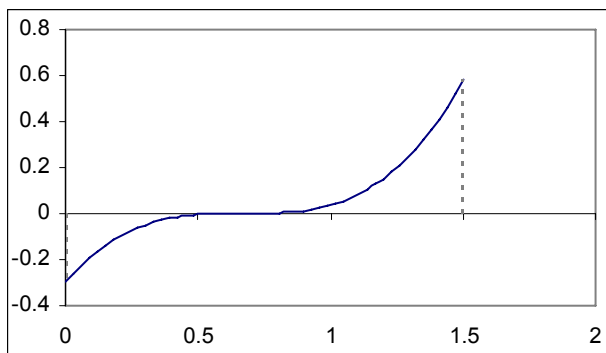
$$f(x) = -\sum_{k=0}^5 (-1)^k \frac{\cos(3(2k+1)x)}{(2k+1)}$$

Presenta una radice $x \approx 0.5236$

Intervallo iniziale $[0, 0.9]$

Punto d'innescio $x_0 = 0.9$

Difficile situazione. Problemi con i metodi che usano le derivate



Test 06

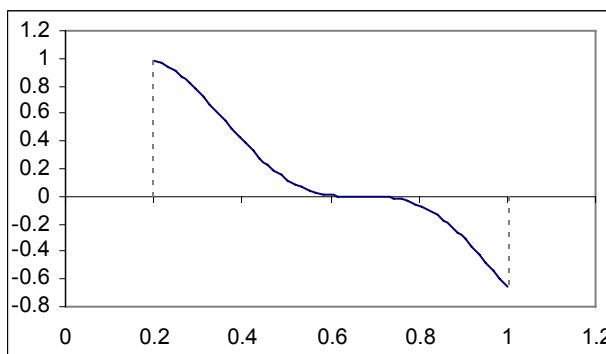
$$f(x) = (x - 2/3)^3$$

Presenta una radice $x = 2/3$

Intervallo iniziale $[0, 1.5]$

Punto d'innescio $x_0 = 0.75$

Le radici multiple mettono in difficoltà quasi tutti i metodi. La precisione della radice è in generale molto minore di quella della funzione



Test 07

$$f(x) = \sin^3(\pi \cdot x - \pi/3)$$

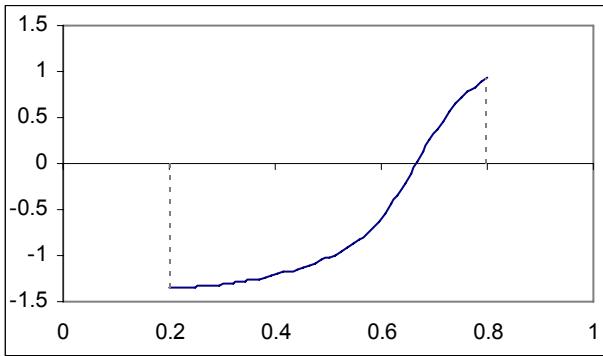
Presenta una radice $x = 2/3$

Intervallo iniziale $[0.2, 1]$

Punto d'innescio $x_0 = 0.4$

Radice multipla

La funzione ha anche altre radici $x = -1/3$, $x = 5/3$, ecc.



Test 08

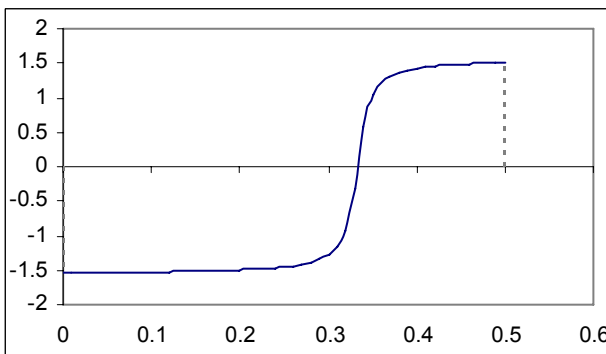
$$f(x) = \arctan(10(x - 2/3)) - \frac{1}{100} \sin(22(x - 2/3))$$

Presenta una radice $x = 2/3$

Intervallo iniziale $[0.2, 0.8]$

Punto d'innesco $x_0 = 0.8$

Funzione fastidiosa per le perturbazioni sulle derivate



Test 09

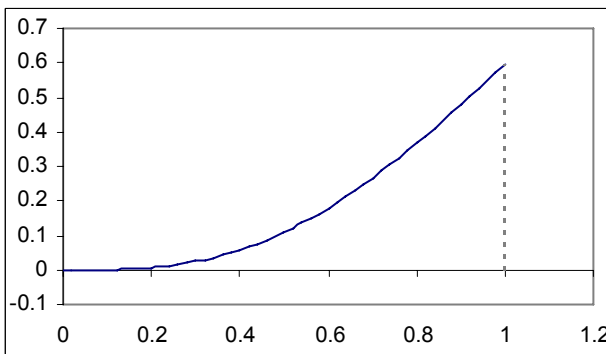
$$f(x) = \arctan(100(x - 2/3))$$

Presenta una radice $x = 2/3$

Intervallo iniziale $[0, 0.5]$

Punto d'innesco $x_0 = 0.5$

Le funzioni "a gradino" sono quasi impossibili per tutti i metodi che usano le derivate. Tutti tranne - sorprendentemente - quello di Halley.



Test 10

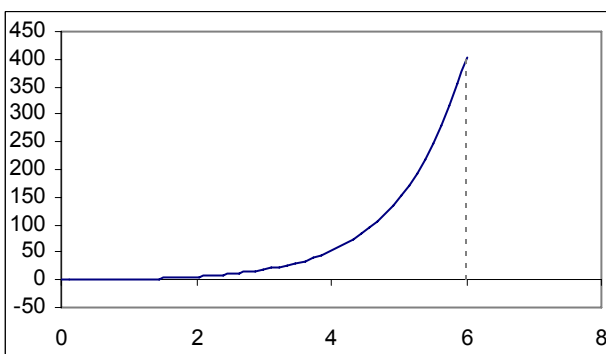
$$f(x) = \sin(x)^3 - (10)^{-3}$$

Presenta una radice $x \approx 0.1001674$

Intervallo iniziale $[0, 1]$

Punto d'innesco $x_0 = 1$

Radice "quasi" multipla



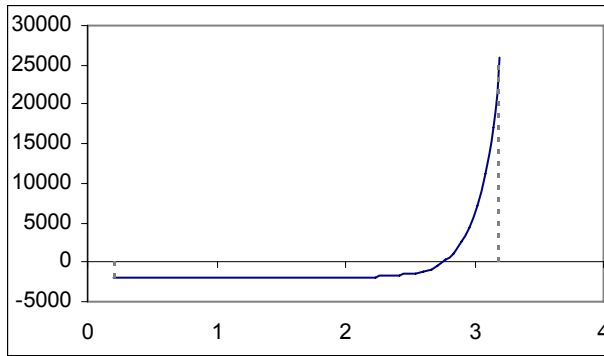
Test 11

$$f(x) = e^x - 2 + \frac{1}{5} \sin(5x)$$

Presenta una radice $x \approx 0.74791$

Intervallo iniziale $[0, 6]$

Punto d'innesco $x_0 = 6$



Test 12

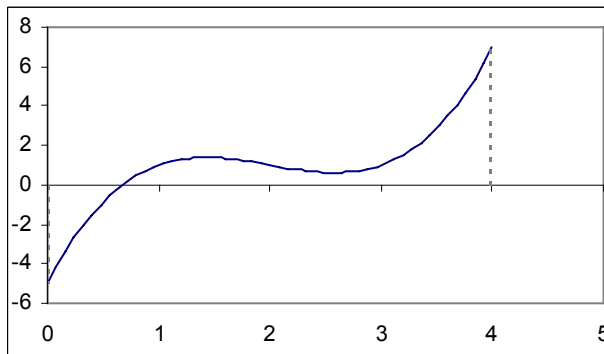
$$f(x) = e^{x^2} - 3000$$

Presenta una radice $x = \sqrt{\log(2000)} \cong 2.7569$

Intervallo iniziale $[0.2, 3.2]$

Punto d'innescio $x_0 = 3.2$

La funzione ha anche una radice negativa simmetrica che può "intrappolare" alcuni metodi o può causare errore di "overflow"



Test 13

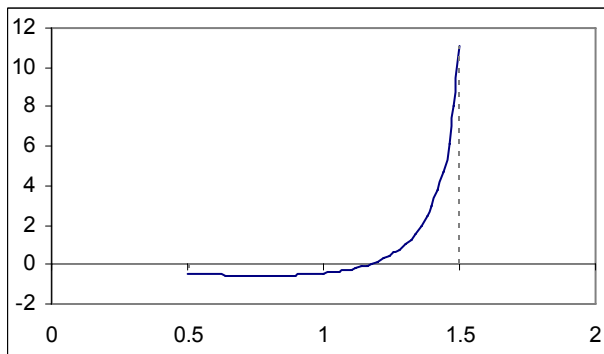
$$f(x) = x^3 - 6x^2 + 11x - 5$$

Presenta una radice $x \cong 0.675282$

Intervallo iniziale $[0, 4]$

Punto d'innescio $x_0 = 4$

Un minimo locale, nell'intervallo di ricerca, riesce ad "ingannare" quasi tutti i metodi. Situazione da evitare.



Test 14

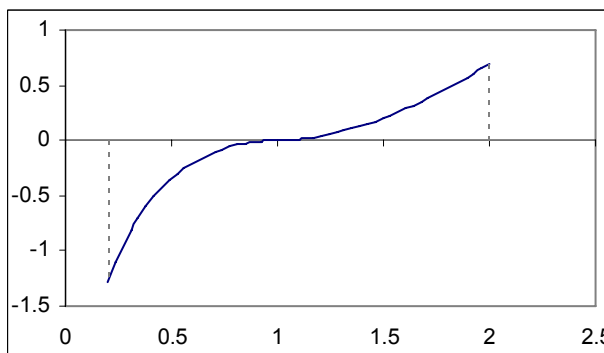
$$f(x) = \tan(x) - 2x$$

Presenta una radice $x \cong 1.16556$

Intervallo iniziale $[0.5, 1.5]$

Punto d'innescio $x_0 = 1.5$

La funzione ha un polo per $x \cong 1.57..$



Test 15

$$f(x) = |x - 1| \cdot \log(x)$$

Presenta una radice $x = 1$

Intervallo iniziale $[0.2, 2]$

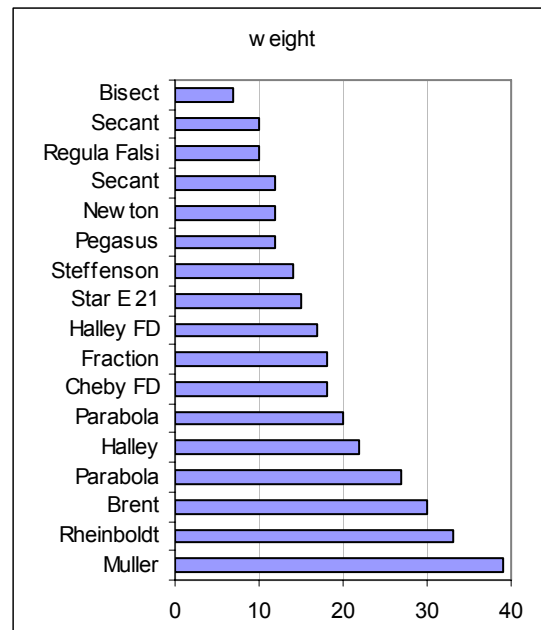
Punto d'innescio $x_0 = 2$

Radice multipla

Costo per iterazione

La seguente tabella riassume il peso o costo per iterazione (weight) dei metodi impiegati nei test. Il costo del calcolo della funzione e delle derivate è stato impostato convenzionalmente pari 5 op.

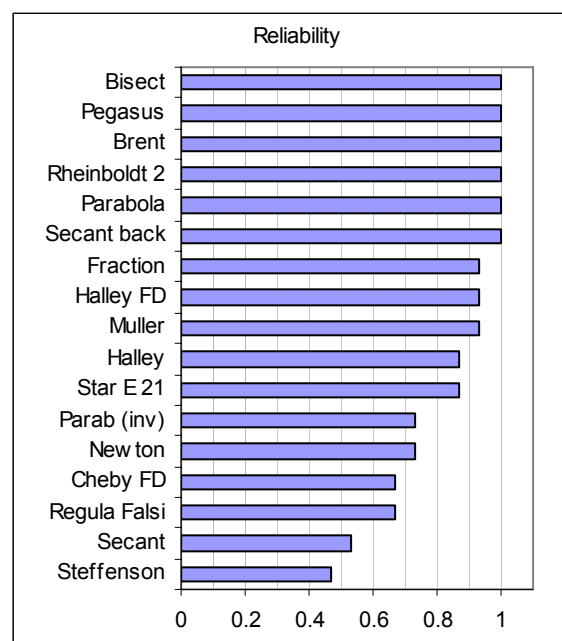
Metodo	weight
Bisect	7
Regula Falsi	10
Secant	10
Pegasus	12
Newton	12
Secant back	12
Steffenson	14
Star E 21	15
Halley FD	17
Cheby FD	18
Fraction	18
Parabola inv.	20
Halley	22
Parabola	27
Brent	30
Rheinboldt 2	33
Muller	39



Classifica Affidabilità

Per "affidabilità" (reliability) abbiamo definito il rapporto fra i successi (goals) e le prove (trials). Per successo del test intendiamo il raggiungimento dell'obiettivo: cioè $|f(x)| < 1E-15$ entro il limite di 400 iterazioni.

Method	Reliability
Bisect	1
Pegasus	1
Brent	1
Rheinboldt 2	1
Parabola	1
Secant back	1
Muller	0.93
Halley FD	0.93
Fraction	0.93
Star E 21	0.87
Halley	0.87
Newton	0.73
Parab (inv)	0.73
Regula Falsi	0.67
Cheby FD	0.67
Secant	0.53
Steffenson	0.47



Primi in classifica risultano, come ci aspettavamo, i cosiddetti metodi "garantiti": Bisezione, Pegasus, Brent, Rheinboldt.

A pari merito, fra i metodi che non hanno mai fallito in questo test risultano anche il metodo delle parabole e il metodo Secant-back-step. Possiamo dire che sono metodi affidabili al 100%.

Potrà sorprendere che un altro metodo garantito matematicamente come la "Regula Falsi" possa invece esibire un indice di affidabilità di circa 67%. Ciò si spiega con il fatto che in alcuni casi il metodo ha superato il limite d'iterazioni impostato e quindi ha fallito l'obiettivo.

Notiamo che intorno al 90% si trovano ben 5 metodi: Halley, Muller, Halley FD, Fraction, Star E21. Il fatto che si tratta di metodi che esibiscono sia qualità di alta affidabilità sia di alta velocità di convergenza rende questi metodi estremamente interessanti. Il metodo di Halley è addirittura del 3° ordine e questo spiega forse la sua "riscoperta" nel calcolo numerico automatico

Il metodo di Newton-Raphson ha mostrato un buon indice di affidabilità, maggiore del 70%, a dispetto dell'enfasi con cui molti testi affermano la sua povera convergenza globale. Anche con i punti d'innescio non "sufficientemente vicini" come raccomandano molti testi, il metodo ha totalizzato 11 goals su 15 tests. Con una scelta ragionevolmente più accurata l'indice sarebbe stato sensibilmente più alto.

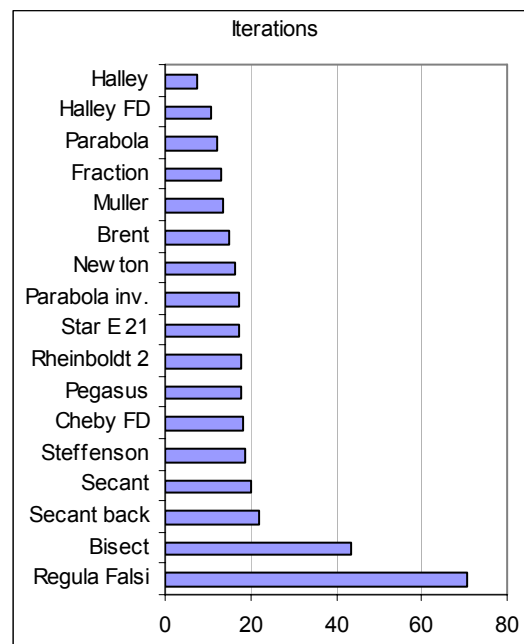
Una più modesta affidabilità, intorno al 50%, hanno mostrato i metodi della secante e di Steffenson.

L'affidabilità è forse l'informazione più importante per chi sviluppa e assembla routine numeriche automatiche.

Classifica Iterazioni

Riporta la media delle iterazioni di ogni metodo^(*).

Method	iterations
Halley	7.6
Halley FD	10.6
Parabola	12.1
Fraction	12.9
Muller	13.8
Brent	14.8
Newton	16.3
Parabola inv.	17.1
Star E 21	17.4
Rheinboldt 2	17.7
Pegasus	18.0
Cheby FD	18.1
Steffenson	18.7
Secant	20.0
Secant back	22.1
Bisect	43.5
Regula Falsi	70.6



(*) La classifica tiene conto solo dei casi di convergenza raggiunta

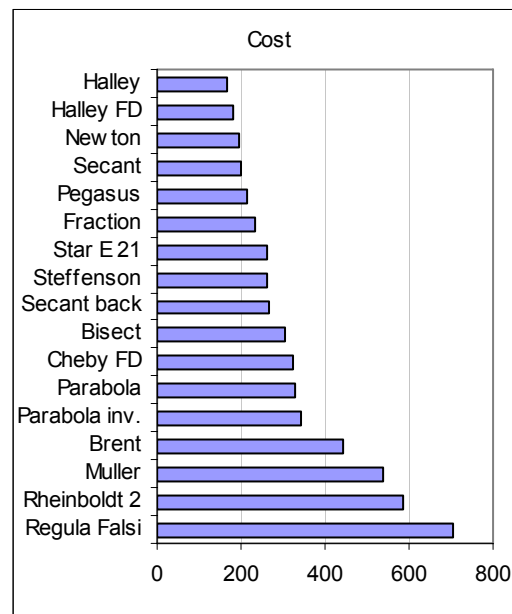
Si tratta di una classifica abbastanza prevedibile: in alto i metodi ad alto ordine e scendendo quelli meno veloci. Più interessante è osservare il profilo: quasi tutti i metodi convergono a 1E-15 in una fascia relativamente stretta di circa 10-20 iterazioni.. Le uniche eccezioni sono il metodo di Bisezione e Regula Falsi: come noto il primo non dipende dalla funzione e converge alla precisione di 1E-15 in circa 50 iterazioni per radici singole; il secondo, in diversi casi, converge molto più lentamente del metodo di bisezione stesso.

Ma in prima approssimazione il numero d'iterazioni per raggiungere la precisione standard di 1E-15 rimane abbastanza contenuto per quasi tutti i metodi. Chiaramente il numero di operazioni per iterazione, cioè il peso (weight) non è uguale per i vari metodi.

Classifica Costo

Il costo è definito, nel caso di convergenza, come numero iterazioni per il peso del metodo. La seguente tabella riporta la classifica del costo medio di ogni metodo (*)

Method	Cost
Halley	168
Halley FD	180
Newton	195
Secant	200
Pegasus	216
Fraction	233
Star E 21	261
Steffenson	262
Secant back	266
Bisect	305
Cheby FD	326
Parabola	328
Parabola inv.	342
Brent	444
Muller	538
Rheinboldt 2	585
Regula Falsi	706



(*) La classifica tiene conto solo dei casi di convergenza raggiunta

Rispetto alla precedente classifica si vedono dei ribaltamenti e delle conferme. Il metodo della secante si trova in 14° posizione nella classifica delle iterazioni (20 iterazioni medie) ma solo 4° in quella del costo effettivi. Grazie infatti al suo basso peso computazionale, il suo costo totale, quando converge, è molto più basso di altri metodi. Viene confermato in prima posizione il metodo di Halley che è risultato quindi efficiente, veloce e affidabile.

Altri metodi molto veloci come quello di Muller, Parabola, e Brent sono stati invece penalizzati dal loro alto peso computazionale.

Notiamo anche che il metodo di bisezione si trova a metà classifica. Grazie al suo bassissimo costo computazione è risultato competitivo con altri metodi teoricamente più veloci.

Classifica Efficienza

Strettamente correlata al costo è l'efficienza del metodo (efficiency), che è inversamente proporzionale al costo stesso. Per confrontare differenti metodi è utile usare l'efficienza relativa dove viene preso come riferimento il numero d'iterazioni del metodo di bisezione. Cioè per un esperimento viene misurato il costo C_B del metodo di bisezione e quello C_A dell'algorithm da confrontare.

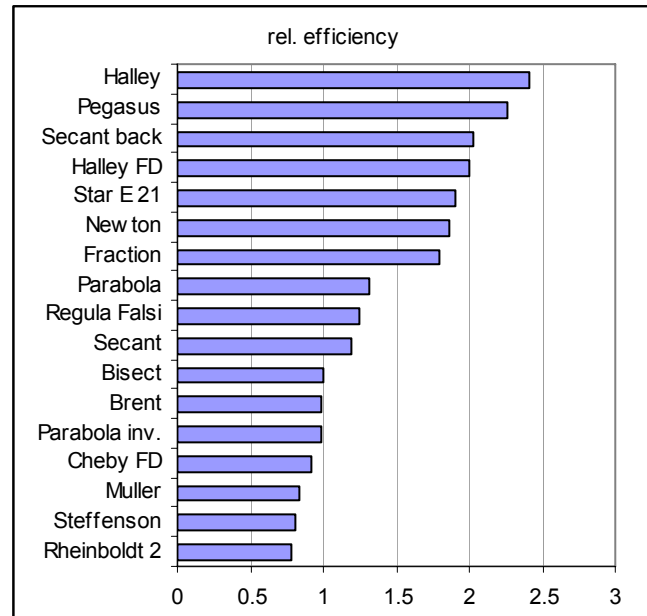
L'efficienza relativa, è:

$$\eta_r = C_B / C_A$$

Esso indica il guadagno di costo computazionale di un algoritmo rootfinding rispetto al metodo di bisezione, considerato convenzionalmente come standard di riferimento.

Nella seguente classifica si sono inseriti anche i casi di non convergenza a cui è stata attribuita efficienza nulla ($\eta_r = 0$) per convenzione.

Method	Efficiency
Halley	2.41
Pegasus	2.26
Secant back	2.03
Halley FD	2
Star E 21	1.91
Newton	1.86
Fraction	1.79
Parabola	1.31
Regula Falsi	1.24
Secant	1.19
Bisect	1
Brent	0.99
Parabola inv.	0.98
Cheby FD	0.92
Muller	0.83
Steffenson	0.81
Rheinboldt 2	0.78



Questa classifica è diversa dalla precedente perchè avendo inserito anche i casi di non convergenza tiene conto oltre all'efficienza anche dell'affidabilità del metodo stesso. E' sicuramente una classifica più aderente ai casi concreti che rispecchia meglio quanto avviene nella pratica attività computazionale.

Il metodo di Steffenson, da metà classifica del costo precipita agli ultimi posti in quella dell'efficienza globale; e il risultato è plausibile in quanto il metodo è effettivamente risultato molto veloce ma anche molto instabile avendo fallito quasi metà dei test.

I più alti valori di efficienza in questo test sono stati riportati sorprendentemente da due metodi, Halley e Pegasus, di caratteristiche molto differenti: Il primo, ad un singolo punto, del 3° ordine, richiede il calcolo della funzione, della derivata prima e seconda, e praticamente estrae ad ogni passo il massimo dell'informazione possibile dalla funzione stessa. Il secondo, a due punti, richiede solo un calcolo della funzione ad ogni passo ed ha un ordine di convergenza medio di circa 1.5. Nonostante queste differenze, i due metodi hanno mostrato in questo test un'efficienza molto simile, sia per quanto l'affidabilità sia per quanto il costo globale. Entrambi hanno guadagnato più del doppio rispetto al metodo di bisezione.

Nella fascia di guadagno fra 1.5 e 2 troviamo ben 5 metodi: Secant back-step, Halley FD, Star, Newton, Fraction. Sicuramente i buoni metodi non mancano.

Il metodo di Brent ha mostrato, paradossalmente, la stessa efficienza del metodo di bisezione. Occorre ripetere che questi risultati comparativi sono validi limitatamente alla precisione standard usata nel test, cioè $1E-15$. Per il raggiungimento di maggiori precisioni probabilmente la classifica dell'efficienza sarebbe sensibilmente diversa in quanto l'ordine di convergenza avrebbe un peso molto maggiore.

I metodo di Rheinbold e di Brent sono entrambi due metodi "ibridi" a convergenza globale garantita. Tuttavia in questo test il metodo di Brent è risultato sensibilmente più efficiente del primo

Zeri di polinomi

La ricerca degli zeri della funzione polinomiale è certamente uno dei campi di ricerca più importanti del calcolo numerico. In generale, un polinomio di grado n , ad una variabile, può essere scritto come:

$$P(z) = a_n z^n + a_{n-1} z^{n-1} + a_{n-2} z^{n-2} \dots + a_2 z^2 + a_1 z + a_0 \quad (1)$$

Per trovare gli zeri reali dei polinomi è possibile usare molti dei metodi descritti per le funzioni reali. Nel caso dei polinomi, però, la ricerca degli zeri si estende a tutto il piano complesso. Inoltre i polinomi presentano alcune proprietà che possono essere utilizzate in modo conveniente per la ricerca degli zeri sia reali che complessi.

Proprietà dei polinomi

- 1) Un polinomio ha sempre un numero di zeri, reali e complessi, pari al suo grado. Si deve contare uno zero tante volte quanto indica la sua molteplicità: uno zero semplice si conta una volta, uno zero doppio si conta due volte, ecc.
- 2) Se i coefficienti sono tutti reali allora gli zeri complessi sono sempre coniugati. Cioè, se esiste una radice $z = a + ib$ allora esiste anche la radice $\bar{z} = a - ib$ e questo permette di ridurre della metà lo sforzo di ricerca degli zeri complessi
- 3) Se i coefficienti sono tutti reali la somma delle radici è pari a: $\sum z_i = -a_{n-1} / a_n$.
- 4) Se i coefficienti sono tutti reali il prodotto delle radici è pari a: $\prod z_i = (-1)^n a_0 / a_n$
- 5) Nota una radice z_1 del polinomio è possibile sempre abbassare il grado del polinomio mediante la divisione $P(z) / (z - z_1) = P_1(z)$. Il polinomio $P_1(z)$, di grado $n-1$, ha le stesse radici di $P(z)$ meno la radice z_1 . Questo procedimento è noto come "deflazione" e permette di ripetere il processo di ricerca delle radici iterativamente. La deflazione in genere introduce degli errori di arrotondamento per cui si deve prendere degli accorgimenti per minimizzare la propagazione di questi errori.
- 6) Nota una radice complessa $z_1 = a - ib$, se i coefficienti sono tutti reali, allora è possibile sempre abbassare il grado del polinomio mediante la divisione:
 $P_1(z) = P(z) / (z - a - ib)(z - a + ib)$. Il polinomio $P_1(z)$, di grado $n-2$, ha le stesse radici di $P(z)$ meno le radici complesse coniugate z_1 e \bar{z}_1 . Questo procedimento è noto come "deflazione complessa" e permette di ripetere il processo di ricerca delle radici iterativamente. Nonostante il nome, la deflazione complessa può essere eseguita completamente con numeri reali essendo $(z - a - ib)(z - a + ib) = (z^2 + 2a z + a^2 + b^2)$
- 7) Se i coefficienti sono tutti interi le eventuali radici intere vanno ricercate fra i divisori esatti del termine noto a_0 . Questa proprietà permette di estrarre dal polinomio le radici intere mediante deflazione esatta, cioè senza arrotondamenti
- 8) Se una radice ha molteplicità m allora è radice anche di tutte le derivate fino a $P^{(m-1)}$

Divisione veloce di polinomi

Il metodo di Ruffini-Horner permette di eseguire la divisione di un polinomio per il binomio $(x - a)$ in modo rapido e sicuro. In generale la divisione restituisce il polinomio quoziente $Q(x)$ ed il resto numerico r , tale che:

$$P(x) = (x - a) \cdot Q(x) + r \tag{1}$$

Esempio

$$(2x^4 + 11x^3 - 53x^2 - 176x + 336)/(x - 1)$$

Costruiamo la tabella seguente

1	2	11	-53	-176	336
1	2	11	-53	-176	336

Il numero 1 (a destra) è il divisore
I coefficienti del polinomio sono nella prima riga, ordinati da sinistra a destra secondo potenze decrescenti

1	2	11	-53	-176	336
1	2	11	-53	-176	336
	2				

Iniziamo l'algorithmo abbassando il 2

1	2	11	-53	-176	336
1	2	11	-53	-176	336
	2	2			
	2	13			

moltiplico il 2 per il numero 1 (divisore) e metto il risultato sotto il 2° coefficiente per fare la loro somma algebrica, che dà 13

1	2	11	-53	-176	336
1	2	11	-53	-176	336
	2	2	13		
	2	13	-40		

moltiplico il 13 per il numero 1 (divisore) e metto il risultato sotto il 3° coefficiente per fare la somma algebrica, che dà -40

1	2	11	-53	-176	336
1	2	11	-53	-176	336
	2	2	13	-40	-216
	2	13	-40	-216	120

Ripetendo il processo fino all'ultimo coefficiente si completa la divisione. I coefficienti del quoziente sono nell'ultima riga, mentre il resto si legge nell'ultimo cella in basso a destra

Quindi : $Q(x) = 2x^3 + 13x^2 - 40x - 216$, $r = 120$

Altri esempi

$$(2x^4 + 11x^3 - 53x^2 - 176x + 336)/(x + 2)$$

$$Q = 2x^3 + 7x^2 - 67x - 42$$
 , $r = 420$

-2	2	11	-53	-176	336
-2	2	11	-53	-176	336
	2	-4	-14	134	84
	2	7	-67	-42	420

$$(2x^4 + 11x^3 - 53x^2 - 176x + 336)/(x + 4)$$

$$Q = 2x^3 + 3x^2 - 65x + 84$$
 , $r = 0$

-4	2	11	-53	-176	336
-4	2	11	-53	-176	336
	2	-8	-12	260	-336
	2	3	-65	84	0

Si noti che, in questo ultimo caso, il resto è zero. Significa che il numero -4 annulla il polinomio $P(x)$ e, di conseguenza, il binomio $(x + 4)$ è un fattore del polinomio stesso

Lo schema di divisione può essere implementato nei programmi in modo molto efficiente. Un esempio è dato dal seguente codice in VB, il cui costo computazionale è: $3n+2$

```
Sub PolyDiv1(a(), b(), r, p)
'Divide il polinomio A(x)= a0+a1*x+a2*x^2+... per (x-p)
'      A(x) = B(x)*(x-p)+R
'A()= Coefficienti di A(x)  A(0)=a0, A(1)=a1, A(2)=a2 ...
'B()= coefficienti di B(x)  B(0)=b0, B(1)=b1, B(2)=b2 ...
'R = resto
  Dim n, j
  n = UBound(a)
  b(n) = 0
  For i = n - 1 To 0 Step -1
    j = i + 1
    b(i) = a(j) + p * b(j)
  Next i
  r = a(0) + p * b(0)
End Sub
```

La divisione del polinomio $P(z) = a_n z^n + a_{n-1} z^{n-1} + a_{n-2} z^{n-2} + \dots + a_2 z^2 + a_1 z + a_0$ per un polinomio quadratico $(z^2 - u z - v)$, tipico della deflazione complessa, può essere implementato mediante il seguente schema iterativo.

$$b_n = 0, \quad b_{n-1} = 0 \quad b_i = a_i + u \cdot b_{i+1} + v \cdot b_{i+2} \quad \text{per } i = n-2 \dots 2, 1, 0, -1, -2$$

dove il polinomio quoziente è: $Q(z) = b_{n-2} z^{n-2} + b_{n-3} z^{n-3} \dots + b_2 z^2 + b_1 z + b_0$
 e il polinomio resto è: $R(z) = b_{-1} z + b_{-2}$

L'algoritmo iterativo può essere implementato agevolmente in forma di tabella su foglio elettronico. Ad esempio, dividere il polinomio: $x^6 - 13x^5 + 106x^4 - 529x^3 + 863x^2 + 744x + 1378$ per il polinomio: $x^2 - 4x + 53$

		a_6	a_5	a_4	a_3	a_2	a_1	a_0
		1	-13	106	-529	863	744	1378
v	u	0	4	-36	68	64	104	0
-53	4	0	0	-53	477	-901	-848	-1378
		1	-9	17	16	26	0	0
		b_4	b_3	b_2	b_1	b_0	b_{-1}	b_{-2}

Tutti i termini della 2° riga, contengono il calcolo di " $u \cdot b_{i+1}$ ", cioè il prodotto di " u " per il valore della 4° riga della colonna precedente.

		a_6	a_5	a_4	a_3
		1	-13	106	-529
v	u	0	4	0	68
-53	4	0	0	-53	477
		1	-9	17	16
		b_4	b_3	b_2	b_1

Tutti i termini della 3° riga, contengono il calcolo di " $v \cdot b_{i+2}$ ", cioè il prodotto di " v " per il valore della 4° riga di due colonne precedenti.

		a_6	a_5	a_4	a_3
		1	-13	106	-529
v	u	0	4	0	68
-53	4	0	0	-53	477
		1	-9	17	16
		b_4	b_3	b_2	b_1

La 4 riga, infine è la somma algebrica dei valori della colonna.

L'algorithmo di divisione polinomiale per un forma quadratica viene implementato in modo efficiente per mezzo di apposite subroutine.

Un esempio è dato dal seguente codice in VB il cui costo computazionale è: 6n

```
Sub PolyDiv2(a(), b(), r1, r0, u, v)
'Divide il polinomio A(x)= a0+a1*x+a2*x^2+.per (x^2-u*x-v)
'      A(x) = B(x)*(x^2-u*x-v) + r1*x+r0
'A()= Coefficienti di A(x)  A(0)=a0, A(1)=a1, A(2)=a2 ...
'B()= coefficienti di B(x)  B(0)=b0, B(1)=b1, B(2)=b2 ...
'r1 = coefficiente di 1° grado del resto
'r0 = termine noto del resto
  Dim n, j, i
  n = UBound(a)
  b(n) = 0
  b(n - 1) = 0
  For i = n - 2 To 0 Step -1
    j = i + 2
    b(i) = a(j) + u * b(i + 1) + v * b(j)
  Next
  r1 = a(1) + u * b(0) + v * b(1)
  r0 = a(0) + v * b(0)
End Sub
```

Calcolo del polinomio e delle sue derivate

La divisione sintetica è un metodo efficace anche per il calcolo numerico dei polinomi.

Infatti, ricordando la relazione

$$P(x) = (x - a) \cdot Q(x) + r$$

Posto $x = a$, si ha

$$P(a) = (a - a) \cdot Q(a) + r \Rightarrow P(a) = r$$

Quindi il resto della divisione sintetica per il numero "a" è il valore del polinomio nel punto $x = a$ stesso. Questo algorithmo per il calcolo del polinomio, conosciuto come algorithmo di Ruffini-Horner, è molto efficiente e stabile. Riprendiamo il polinomio precedente e calcoliamo il valore per $x = -2$, con il metodo classico della sostituzione e con la divisione sintetica

metodo classico

$$\begin{aligned}
 P(x) &= 2x^4 + 11x^3 - 53x^2 - 176x + 336 \\
 P(-2) &= 2(-2)^4 + 11(-2)^3 - 53(-2)^2 - 176(-2) + 336 = \\
 &= 2 \cdot 16 + 11(-8) - 53 \cdot 4 - 176(-2) + 336 = \\
 &= 32 - 88 - 212 - 352 + 336 = 420
 \end{aligned}$$

metodo Ruffini-Horner

	2	11	-53	-176		336
-2		-4	-14	134		84
	2	7	-67	-42		420

E' chiara la superiorità del metodo della divisione, sia dal punto della laboriosità sia dal punto della precisione del risultato. Il motivo è che la divisione sintetica evita il calcolo diretto delle potenze che diventano grandi man mano che aumenta il valore sostituito (e con esso aumenta anche la probabilità di errore o di overflow).

L seguente Function VB implementa in modo efficiente il calcolo del polinomio

```
Function Poly_Eval(x, A())
'Calcola il polinomio A(x)= a0+a1*x+a2*x^2+...an*x^n nel punto x
Dim n, y, i
  n = UBound(A)
  y = A(n)
  For i = n - 1 To 0 Step -1
    y = A(i) + y * x
  Next
  Poly_Eval = y
End Function
```

Il costo computazionale del calcolo del polinomio di grado n è: $2n$

Il seguente codice implementa invece il calcolo contemporaneo del polinomio e di tutte le sue n derivate

```
Sub DPolyn_Eval(x, A, D)
'calcola il polinomio e tutte le sue derivate nel punto x
'D(0) = A(x) , D(1) = A'(x), D(2) = A''(x) ...
'A()= Coefficienti di A(x)  A(0)=a0, A(1)=a1, A(2)=a2 ...
'D()= valori calcolati
Dim n, y, i, k
  n = UBound(A)
  For k = n To 0 Step -1
    y = A(k)
    For i = 0 To k
      D(i) = D(i) * x + y
      y = y * (k - i)
    Next i
  Next k
End Sub
```

Al termine nel vettore D si trovano i valori del polinomio e delle sue derivate calcolate nel punto x

$$D(0) = P(x), D(1) = P^{(1)}(x), D(2) = P^{(2)}(x), D(3) = P^{(3)}(x), \dots D(n) = P^{(n)}(x),$$

Il costo computazionale del calcolo del polinomio di grado n e di tutte le sue derivate è: $3(n^2+3n+2)$

Calcolo di polinomi in precisione finita

Consideriamo il polinomio di 10° grado avente le radici: $x = 1$ ($m = 9$), $x = 2$ ($m = 1$), dove con m si è indicata la molteplicità.

Il polinomio può essere scritto e calcolato con 3 forme diverse

- | | Formula |
|----|---|
| 1) | $(x-2)(x-1)^9$ |
| 2) | $2-19x+81x^2-204x^3+336x^4-378x^5+294x^6-156x^7+54x^8-11x^9+x^{10}$ |
| 3) | $((((((((x-11)x+54)x-156)x+294)-378)x+336)x-204)x+81)x-19)x+2$ |

La prima è la forma fattorizzata, che è in genere la forma a cui tendere per avere una forma più compatta. Necessita della conoscenza di tutte le radici del polinomio.

La seconda è la nota forma somma di potenze ottenibile attraverso i coefficienti del polinomio. Si tratta in genere della forma più comune.

La terza infine è la cosiddetta forma di Horner, ottenibile dai coefficienti come la precedente ma priva del calcolo delle potenze. Il metodo iterativo Ruffini-Horner è identico numericamente a questa formula. Dal punto di vista algebrico queste forme sono equivalenti. Non così invece dal punto di vista del calcolo numerico.

Molti algoritmi di ricerca delle radici, sfruttano la conoscenza del valore del polinomio (e/o delle sue derivate) per ottenere iterativamente approssimazioni sempre migliori delle sue radici.

Non potendo conoscere direttamente le radici, di solito la forma (3) è quella più comunemente usata - grazie alla sua efficienza - per il calcolo dei polinomi reali e/o complessi.

Vediamo come si comportano le diverse formule nel caso pratico. Supponiamo di calcolare il polinomio in un punto prossimo alla radice multipla usando l'aritmetica standard dei comuni PC a 32bit (15 cifre significative).

Calcolo per $x = 1.0822 \Rightarrow P(x) = -1.5725136038295733584E-10$ ⁽¹⁾

Formula	P(x) ²	Rel. Error
1)	-1.57251360383E-10	7.8904E-15
2)	-1.57207580287E-10	0.00027849
3)	-1.57209356644E-10	0.00026718

Come si vede, gli schemi (2) e (3) approssimano il valore con un errore relativo di circa 1E0-4 (0.1%), mentre la formula (1) dà un valore praticamente perfetto. Negli ultimi due casi solo le prime 4 cifre sono corrette.

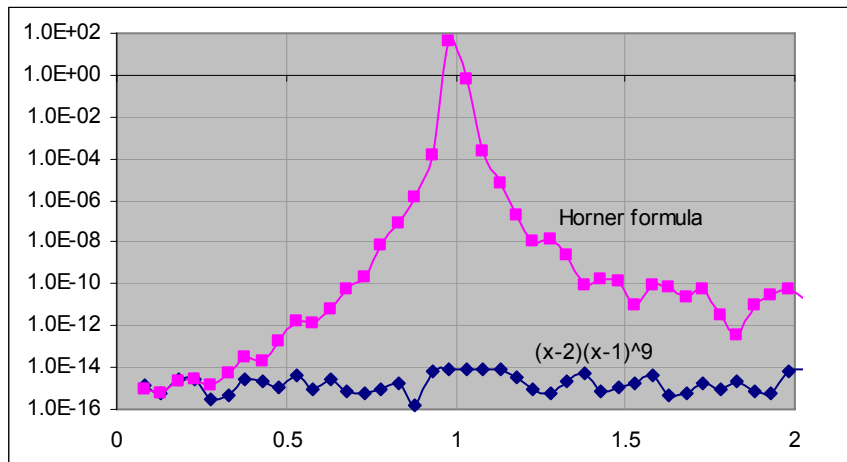
E' quindi intuibile aspettarsi comportamenti molto diversi nel caso che l'algoritmo chiami la formula (1) anziché le formule (2) o (3). Come già detto, purtroppo la formula (1) non è utilizzabile per la mancanza di conoscenza delle radici essendo, appunto, l'oggetto della ricerca. Di solito è usata da diversi autori - per comodità - nei test degli algoritmi. Tuttavia, come abbiamo visto, la risposta sembra irrealisticamente troppo "perfetta" per dare attendibilità al test.

Il calcolo numerico di un "vero" polinomio, cioè di un polinomio noto dai suoi coefficienti, è molto diverso da quello del polinomio fattorizzato, a causa degli errori di arrotondamento legati alla precisione finita del calcolo.

¹ valore calcolato con aritmetica estesa a 30 cifre

² Le cifre in nero sono esatte

Nel grafico seguente è stato riportato l'andamento dell'errore relativo ottenuto con la formula fattorizzata (1) e con la formula di Horner (3) in un intorno della radice $x = 1$



Il grafico mostra che la differenza fra le due formule di calcolo è rilevante proprio in vicinanza della radice. Questo dimostra che, per avvicinarsi il più possibile alle condizioni reali d'uso, gli algoritmi di ricerca delle radici dovrebbero essere provati nella condizione peggiore, cioè con (2) o (3). Per contro, quando sono note le radici, il calcolo più accurato ed efficiente è quello che usa la formula fattorizzata

Effetto barriera

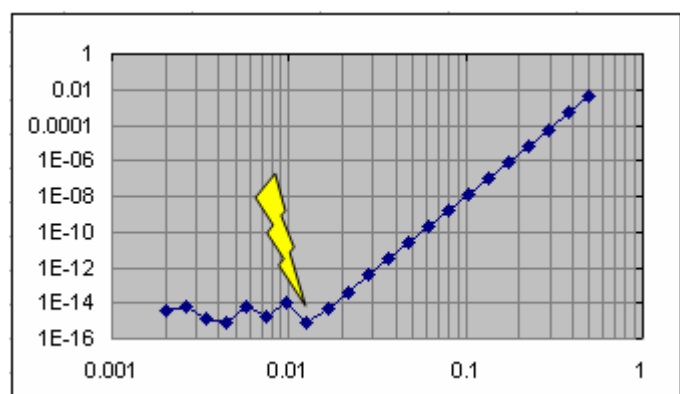
Come abbiamo visto, il calcolo con precisione finita di un polinomio sia nella forma di Horner sia in quella normale è affetto da errori. Tali errori diventano ancor più fastidiosi in presenza di radici multiple perchè di fatti limitano l'accuratezza con cui la radice può essere approssimata: maggiore è la molteplicità è maggiore è la distanza a cui si può avvicinare la radice. Tale fenomeno è conosciuto come barriera limite inferiore (underflow breakdown).

Ad esempio supponiamo di avere il polinomio di 8° grado con l'unica radice $x = 1$, molteplicità 8

$$P(x) = x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1$$

Calcoliamo, usando aritmetica a 15 cifre, il polinomio nel punto $x = 1+dx$, dove "dx" è un valore progressivamente sempre più piccolo. In tal modo simuliamo l'avvicinamento di un algoritmo rootfinder.

Per ogni punto campionato rileviamo il valore $|P(x)|$ e riportiamo la coppia di valori $(dx, |P(x)|)$ in un grafico bi-logaritmico. Quello che osserviamo è riportato in figura



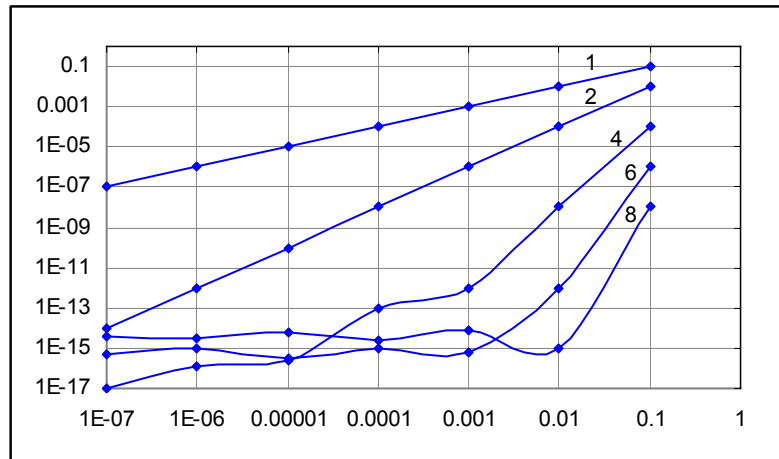
Come si vede in prossimità del punto $dx = 0.01$, la curva cessa di essere rettilinea ed assume un andamento caotico. Significa che siamo arrivati al limite minimo di calcolo

Gli errori di arrotondamento prodotti nel calcolo di $P(x)$ per $dx < 0.01$ sono preponderanti e mascherano completamente il vero valore del polinomio. Nessun algoritmo di approssimazione

della radice può andare oltre questo limite attraverso il semplice calcolo del polinomio dato. E' di fatto una "barriera" di precisione oltre il quale non si può scendere.

Questo limite dipende da vari fattori: principalmente dalla precisione di macchina (nei comuni PC è di 15 cifre); poi dalla molteplicità della radice, dal grado del polinomio e come abbiamo visto anche dalla formula usata per il calcolo del polinomio stesso

Il grafico seguente riporta gli stessi grafici per le molteplicità di $m = 1, 2, 4, 6, 8$



Per le molteplicità di $m \leq 2$ non ci sono problemi a raggiungere precisioni inferiori a $1E-7$, ma per molteplicità 4 la barriera è di circa $1E-5$ e si sposta a 0.001 per $m = 6$ e 0.01 per $m = 8$.

Per superare la barriera si deve usare o l'aritmetica in multiprecisione o adottare altri metodi che evitano il calcolo diretto del polinomio originale (vedi par. "Eliminazione della molteplicità").

Deflazione

Quando di un polinomio si conosce una radice, reale o complessa, si può eliminarla dal polinomio per mezzo della divisione sintetica per ottenere un polinomio ridotto di grado inferiore.

Dato il polinomio

$$16x^6 - 58x^5 - 169x^4 + 326x^3 - 230x^2 + 384x - 45$$

Assumiamo di aver trovato, in qualche modo, la radice reale $x = -3$ e le radici complesse coniugate $x = \pm i$. Eliminiamo per prima la radice reale, dividendo per $(x+3)$

-3	16	-58	-169	326	-230	384	-45
		-48	318	-447	363	-399	45
	16	-106	149	-121	133	-15	0

le radici complesse coniugate $a+ib$ possono essere eliminate dividendo per il fattore quadratico $x^2 - u \cdot x - v$, dove $u = 2 \cdot a$ e $v = -(a^2 + b^2)$.

			16	-106	149	-121	133	-15
v	u			0	0	0	0	0
-1	0			0	-16	106	-133	15
			16	-106	133	-15	0	0

Quindi il polinomio $16x^3 - 106x^2 + 133x - 15$ contiene tutte le radici del polinomio originale meno quelle eliminate.

Questo processo, chiamato "deflazione" o "riduzione" è molto comune negli algoritmi rootfinding sequenziali. Quando una radice viene trovata, il polinomio viene ridotto di grado per deflazione e il processo viene riapplicato per la ricerca di una nuova radice e così via fino ad ottenere un polinomio di primo o di secondo grado. Quest'ultimi vengono risolti normalmente per mezzo delle specifiche formule.

Deflazione con decimali

Se tutti i coefficienti e tutte le radici hanno valori interi la deflazione è sempre esatta. Al contrario se una o più radice sono decimali o anche se le radici trovate sono affette da errori di approssimazione, il processo di deflazione propaga in cascata e, talvolta, amplifica questi errori. Le ultime radici trovate possono accumulare quindi molti errori ed essere sensibilmente diverse da quelle esatte.

Ad esempio, dato polinomio $x^5 - 145x^4 + 4955x^3 - 46775x^2 + 128364x - 86400$

le radici esatte sono 1, 3, 9, 32, 100, proviamo ad effettuare la deflazione introducendo un errore relativo di approssimazione verosimile di $1E-6$, (0.0001 %) per le prime 4 radici . Facciamo la deflazione in ordine crescente partendo dalla radice più piccola.

1.000001	1	-145	4955	-46775	128364	-86400
	0	1.000001	-144.0001	4811.005	-41964.04	86400.05
	1	-144	4811	-41964	86399.96	0.049104
3.000003	1	-144	4811	-41964	86399.96	
	0	3.000003	-423.0004	13164.01	-86400.04	
	1	-141	4387.999	-28799.98	-0.075186	
9.000009	1	-141	4387.999	-28799.98		
	0	9.000009	-1188.001	28800.01		
	1	-132	3199.998	0.030345		
32.00003	1	-132	3199.998			
	0	32.00003	-3200.002			
	1	-99.99996	-0.003385			

Per vedere come si comporta l' algoritmo prendiamo l'ultimo risultato della deflazione che è il polinomio di primo grado e risolviamolo: $x - 99.99996 = 0 \Rightarrow x = 99.99996$ (err. rel. = $4.5E-07$)
 Come si vede, l'errore è comparabile con quelli introdotti nelle radici per cui l' algoritmo ha limitato la propagazione. Non così, invece se partiamo dalla radice più grande in ordine decrescente

100.0001	1	-145	4955	-46775	128364	-86400
	0	100.0001	-4499.99	45500.6	-127441	92342.36
	1	-44.9999	455.0055	-1274.4	923.4227	5942.364
32.00003	1	-44.9999	455.0055	-1274.4	923.4227	
	0	32.00003	-415.996	1248.299	-835.373	
	1	-12.9999	39.00931	-26.1054	88.04928	
9.000009	1	-12.9999	39.00931	-26.1054		
	0	9.000009	-35.9988	27.0949		
	1	-3.99986	3.010541	0.989503		
3.000003	1	-3.99986	3.010541			
	0	3.000003	-2.99957			
	1	-0.99986	0.01097			

Risolvendo l'ultimo polinomio si ha $x - 0.99986 = 0 \Rightarrow x = 0.99986$ (err. rel. = $1.4E-04$)
 In questo caso l'errore finale è molto maggiore quello introdotto. Per cui l' algoritmo ha amplificato l'errore più di 100 volte.
 Questo risultato è generalizzabile: l' algoritmo di deflazione crescente tende ad essere stabile mentre quello decrescente tende ad amplificare gli errori.
 Questo suggerisce la convenienza di iniziare la ricerca dalle radici di minimo modulo. Purtroppo questo non sempre è possibile. Molti algoritmi ricercano le radici in modo random ovvero partendo da quelle di massimo modulo, perchè più facilmente rintracciabili.

Deflazione reciproca

Per ovviare all'instabilità nel caso che le radici si succedano in ordine decrescente è possibile ricorrere alla variante del polinomio reciproco.

Ad esempio, trovata la prima radice $x = 100 \pm 1E-6$, s'inverte $z = 1/x$ e si riduce il polinomio reciproco $86400x^5 - 128364x^4 + 46775x^3 - 4955x^2 + 145x - 1$ anzichè quello originale.

L'accorgimento si ripete per ogni radice trovata nell'ordine: 32, 9, 3

Lo schema di deflazione reciproca è quindi

x	1/x	86400	-128364	46775	-4955	145	-1
100.0001	0.01	0	863.999136	-1274.999	454.9996	-44.99996	0.999999
		86400	-127500	45500.001	-4500	100	-5.94E-07
x	1/x	86400	-127500	45500.001	-4500	100	
32.000032	0.03125	0	2699.9973	-3899.996	1299.999	-99.99995	
		86400	-124800	41600.005	-3200.002	9.11E-05	
x	1/x	86400	-124800	41600.005	-3200.002		
9.000009	0.111111	0	9599.9904	-12799.99	3199.999		
		86400	-115200.01	28800.016	-0.002963		
x	1/x	86400	-115200.01	28800.016			
3.000003	0.333333	0	28799.9712	-28799.99			
		86400	-86400.042	0.0312043			

Risolvendo l'ultimo polinomio si ha $86400 \cdot x - 86400.042 = 0 \Rightarrow x = 1.0000005$ (err. rel. = $5E-07$)

Come si vede, l'errore è comparabile con quelli introdotti nelle radici precedenti per cui l'algoritmo di deflazione reciproca è stabile per radici in ordine decrescente.

Traslazione

La traslazione di variabile è un'operazione molto frequente e viene usata per scopi diversi: in generale per semplificare la forma di un polinomio, per rendere più semplici i calcoli, per scomporre in fattori, ecc.

Ad esempio si può rendere centrato un polinomio rispetto alle sue radici mediante una traslazione

Il polinomio si dice centrato, o baricentrico, se la somma algebrica delle sue radici è nulla.

In tal caso anche il suo coefficiente a_{n-1} deve essere nullo. Infatti è:

$$\sum_{i=1}^n x_i = 0 \Leftrightarrow a_{n-1} = -\sum_{i=1}^n x_i$$

Qualunque polinomio può essere centrato effettuando la trasformazione $x = z - \frac{a_{n-1}}{n \cdot a_n}$

Esempio

$$P(x) = x^3 + 9x^2 + 28x + 30$$

posto $x = z - 3$

$$P(z) = (z - 6)^3 + 9(z - 6)^2 + 28(z - 6) + 30$$

Da cui svolgendo le potenze e sommando si ha:

$$P(z) = (z^3 - 9z^2 + 27z - 27) + (9z^2 - 54z + 81) + (28z - 84) + 30 = z^3 + z$$

Come si nota il polinomio trasformato in z è molto più semplice di quello originale tanto da poter essere fattorizzato semplicemente per raccoglimento

$$P(z) = z^3 + z = z(z^2 + 1)$$

Le radici di $P(x)$ sono quindi

$$z(z^2 + 1) = 0 \Rightarrow z = \{0, \pm i\} \Rightarrow z = x + 3 \Rightarrow x = \{3, 3 \pm i\}$$

La centratura del polinomio è utile in generale anche per ridurre i coefficienti e semplificare i calcoli. A questo scopo non è necessario la centratura completa; basta "avvicinarsi" al polinomio centrato. Questo è utile talvolta per evitare i numeri decimali. Ad esempio.

$$P(x) = x^4 - 22x^3 + 179x^2 - 638x + 840$$

Cerchiamo di ridurre i coefficienti mediante traslazione. Tuttavia la trasformazione esatta $z = x - (-22)/4 = x + 5.5$ introdurrebbe dei valori decimali, per cui ci accontentiamo di effettuare una centratura approssimata con la trasformazione $z = x + 5$.

$$P(x) = (x + 5)^4 - 22(x + 5)^3 + 179(x + 5)^2 - 638(x + 5) + 840$$

Da cui svolgendo le potenze e sommando si ha il polinomio semplificato in z :

$$P(z) = z^4 - 2z^3 - z^2 + 2z$$

Come si vede la traslazione è una tecnica molto potente di semplificazione dei polinomi. Tuttavia viene in genere evitata a causa della laboriosità degli sviluppi delle potenze.

E' possibile evitare lo sviluppo diretto delle potenze per mezzo delle seguenti formule

$$b_k = P^{(k)}(a) / k! \quad \text{per } k = 0, 1, 2, \dots, n$$

dove b_k sono i coefficienti del polinomio traslato $P(x+a)$

La formula precedente evita lo sviluppo delle potenze ma necessita del calcolo dei fattoriali. Ma usando ancora la divisione sintetica si può ottenendo direttamente i coefficienti del polinomio traslato in modo ancora più efficiente

Divisione sintetica iterativa

Si tratta di applicare iterativamente l'algoritmo di divisione sintetica di Ruffini-Horner al polinomio originale, al polinomio quoziente, al quoziente successivo, e così via, fino al termine. Il numero divisore "a" è la quantità di cui vogliamo traslare $x = z + a$

Vediamo subito un esempio pratico, con il polinomio precedente

$$P(x) = x^4 - 22 \cdot x^3 + 179 \cdot x^2 - 638 \cdot x + 840$$

Vogliamo effettuare la trasformazione di variabile $z = x + 5$ per ottenere il nuovo polinomio

$$P(z) = b_4 z^4 + b_3 z^3 + b_2 z^2 + b_1 z + b_0$$

Il primo coefficiente è sempre uguale al primo coefficiente del polinomio originale, cioè $b_4 = a_4 = 1$
 Gli altri coefficienti b_0, b_1, b_2, b_3 si trovano con la divisione sintetica iterativa

	1	-22	179	-638	840
5		5	-85	470	-840
	1	-17	94	-168	0
		5	-60	170	
	1	-12	34	2	
		5	-35		
	1	-7	-1		
		5			
	1	-2			

nella 1° riga si scrivono i coefficienti del polinomio originale. A sinistra, la quantità da traslare che è 5 nel nostro caso.

Nella 3° riga si ottengono i coefficienti del quoziente Q_1 e il resto (in giallo).

Si ripete la divisione partendo dai coefficienti del quoziente Q_1 e si ottiene il quoziente Q_2 nella 5° riga e un nuovo resto.

Si ripete fino ad avere un solo coefficiente

I coefficienti b_0, b_1, b_2, b_3 sono i resti di tutte le divisioni sintetiche nell'ordine crescente

Quindi il polinomio traslato è

$$P(z) = z^4 - 2 \cdot z^3 - z^2 + 2 \cdot z$$

che avevamo già ottenuto nel paragrafo precedente

Si noti la compattezza e la facilità di questo metodo rispetto all'espansione classica del polinomio. Questo metodo ha validità generale e può essere di grande aiuto nella manipolazione di polinomi, specialmente di quelli di più alto grado

Esempio. Trovare le radici del polinomio di 4° grado

$$P(x) = 5 \cdot x^4 - 120 \cdot x^3 + 919 \cdot x^2 - 2388 \cdot x + 1980$$

Rendiamo centrato il polinomio mediante traslazione $x = z - \frac{a_{n-1}}{n \cdot a_n} \Rightarrow x = z + 6$

Il primo coefficiente $b_4 = 5$; gli altri si trovano con la divisione iterativa

	5	-120	919	-2388	1980	
6		30	-540	2274	-684	
	5	-90	379	-114	1296	$\Rightarrow b_0 = 1296$
		30	-360	114		
	5	-60	19	0		$\Rightarrow b_1 = 0$
		30	-180			
	5	-30	-161			$\Rightarrow b_2 = -161$
		30				
	5	0				$\Rightarrow b_3 = 0$

Quindi il polinomio traslato è

$$P(z) = 5 \cdot z^4 - 161 \cdot z^2 + 1296$$

Che è un polinomio biquadratico, le cui radici sono

$$\Delta = 161^2 - 4 \cdot 5 \cdot 1296 = 1 \Rightarrow z^2 = \frac{161 \pm 1}{10} \Rightarrow z^2 = \{16, 81/5\} \Rightarrow x = \{\pm 4, \pm 9\sqrt{5}/5\}$$

Il seguente codice effettua la traslazione del polinomio in $P(x + x_0)$

```
Sub PolyShift(x0, A, B)
'Trasla il polinomio p(x) in p(x+x0)
'A()= Coefficienti del polinomio originale A(x)=[a0, a1, a2 ...]
'B()= Coefficienti del polinomio traslato B(x)=[b0, b1, b2 ...]
Dim n, i, k, y
n = UBound(A)
For i = 0 To n
    B(i) = A(i)
Next i
For k = 0 To n
    y = 0
    For i = n To k Step -1
        y = y * x0 + B(i)
        B(i) = B(i) * (i - k) / (k + 1)
    Next i
    B(k) = y
Next k
End Sub
```

Costo computazionale. se il grado del polinomio è n, il costo per la traslazione è $3(n^2+3n+2)$

Ad esempio, per un polinomio di 5° grado il costo è 126 op

Ricerca dell'area delle radici

Per il funzionamento della maggior parte degli algoritmi di approssimazione delle radici è necessario avere almeno una conoscenza anche vaga della loro posizione nel piano complesso. Per una radice reali sarebbe auspicabile avere un intervallo di segregazione (bracketing). Per una radice complessa almeno un'area del piano che la contiene.

Metodo grafico

La ricerca delle radici reali di un polinomio si può fare per via grafica analogamente a quanto già detto per le funzioni reali $f(x)$. La ricerca delle radici complesse invece ha bisogno di un differente approccio, più complicato, come è facile intuire.

Ad esempio vogliamo isolare graficamente tutte le radici del polinomio $z^3 - z^2 + 2z - 2$

Facciamo la sostituzione di variabile: $z = x + iy$.

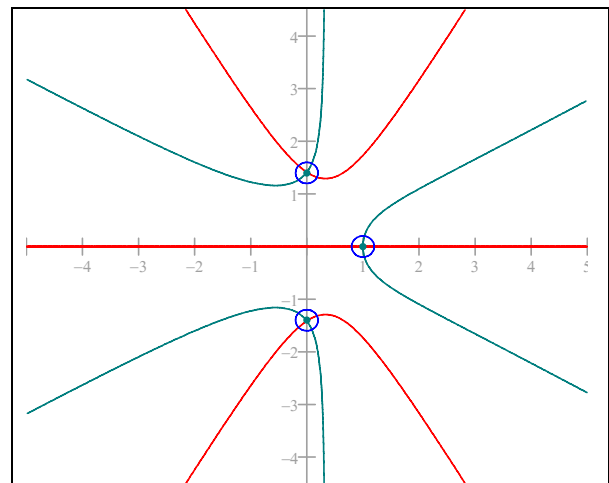
$$(x + iy)^3 - (x + iy)^2 + 2(x + iy) - 2 = [x^3 - x^2 + x(2 - 3y^2) + y^2 - 2] + i[y(3x^2 - 2x - y^2 + 2)]$$

Poniamo

$$\text{Re}(x,y) = x^3 - x^2 + x(2 - 3y^2) + y^2 - 2 = 0 \tag{1}$$

$$\text{Im}(x,y) = y(3x^2 - 2x - y^2 + 2) = 0 \tag{2}$$

Ora si tratta di tracciare il grafico (x,y) delle funzioni implicite definite da (1) e (2). Questo può essere fatto per mezzo di appositi programmi matematici di grafica evoluta facilmente reperibili anche nel pubblico dominio¹. La figura mostra il grafico della curva $\text{Re}(x,y) = 0$ (blu) sovrapposta alla curva di equazione $\text{Im}(x,y) = 0$ (rossa). Si noti che l'asse x è una soluzione della parte immaginaria. Le radici, cioè i punti d'intersezioni, sono chiaramente individuabili:
Le radici si trovano nell'intorno dei punti:



$$P_i = (1, 0), (0, 1.4), (0, -1.4)$$

Il metodo grafico, ovviamente non può essere utilizzato dai programmi automatici ed inoltre risulta un po' laborioso, specialmente per lo sviluppo delle potenze di alto grado ma i risultati sono sempre eccellenti sia per la globalità sia per la precisione. Quando possibile raccomandiamo la sua applicazione.

In allegato sono riportati gli sviluppi complessi delle prime 9 potenze di $(x + iy)$

¹ Il grafico riportato è stato ottenuto con WinPlot, un ottimo programma gratuito di grafica matematica della Peanut Software reperibile in Internet. Ne esiste anche una versione in italiano. Ovviamente vanno bene anche altri eccellenti programmi di grafica matematica.

Metodo della matrice compagna

Un metodo generale per la limitazione di tutte le radici di un polinomio, riportato in tutti i testi di matematica, si basa sulla matrice compagna.

Dato un polinomio normalizzato (cioè con $a_n = 1$) si compone la seguente matrice

$$a(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} + x^n$$

$$A = \begin{bmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -a_{n-1} \end{bmatrix}$$

Gli autovalori di A sono, come noto, le radici del polinomio. Applicando il teorema di Gershgorin per righe, si hanno le seguenti relazioni

$$|\lambda| \leq |a_0| \quad |\lambda| \leq |a_i| + 1 \quad |\lambda - a_{n-1}| \leq 1 \quad \text{per } i = 1, 2, n-2$$

Applicando il teorema alle colonne si hanno le seguenti relazioni

$$|\lambda| \leq 1 \quad |\lambda - a_{n-1}| \leq \sum_{i=0}^{n-2} |a_i|$$

Tutte queste relazioni si possono riassumere nella seguente formula

$$|x| = \max \left\{ 1, |a_0|, \sum_{i=0}^{n-1} |a_i|, 1 + \max_{i=1}^{n-1} |a_i| \right\} \quad (1)$$

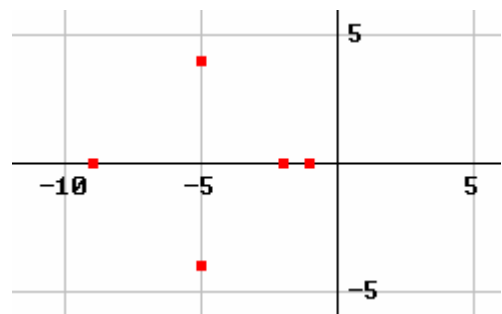
Questa formula è certamente valida dal punto di vista teorico ma dal punto di vista numerico è poco usata in quanto solitamente è troppo poco restrittiva. Vediamo il seguente esempio

$$x^5 + 22 \cdot x^4 + 190 \cdot x^3 + 800 \cdot x^2 + 1369 \cdot x + 738$$

Le radici sono

$$x = -5 - 4 \cdot i, x = -5 + 4 \cdot i, x = -9, x = -2, x = -1$$

La (1) fornisce $|x| < \{1, 738, 3119, 1370\} = 3119$



La relazione precedente rappresenta geometricamente un cerchio centrato nell'origine di raggio $\rho = 3119$; chiaramente è una limitazione troppo ampia.

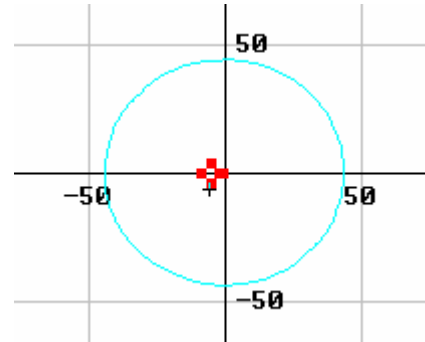
Metodo del cerchio delle radici

Questo metodo si basa sulla seguente formula valida per un polinomio normalizzato (cioè con $a_n = 1$)

$$\rho = 2 \max \{ |a_{n-k}|^{1/k} \} \text{ per } k = 1, 2, \dots, n \quad (2)$$

Applicata al polinomio dell'esempio precedente dà come raggio $\rho = 44$, che è ampio ma molto migliore comunque della limitazione ottenuta con il metodo della matrice compagna

Un miglioramento sensibile si può ottenere osservando che la (2) funziona bene se a_{n-1} è piccolo, rispetto agli altri coefficienti



Variante del polinomio centrato

Per rendere il coefficiente a_{n-1} nullo o quasi nullo si può ricorrere sempre alla traslazione per rendere il polinomio centrato (o quasi centrato) rispetto al baricentro delle sue radici che è dato da

$$b = \frac{1}{n} \sum x_i = -\frac{a_{n-1}}{n a_n} \quad (3)$$

Nel caso del polinomio $x^5 + 22x^4 + 190x^3 + 800x^2 + 1369x + 738$

il centro si trova in $C = (-22/5, 0) = (-4.4, 0)$.

Arrotondiamo 4.4 all'intero per evitare i coefficienti decimali e quindi facciamo la traslazione per mezzo della sostituzione di variabile $x = z - 4$

L'operazione di traslazione viene effettuata con il metodo della divisione iterativa precedentemente visto.

	1	22	190	800	1369	738
-4		-4	-72	-472	-1312	-228
	1	18	118	328	57	510
		-4	-56	-248	-320	
	1	14	62	80	-263	
		-4	-40	-88		
	1	10	22	-8		
		-4	-24			
	1	6	-2			
		-4				
	1	2				

Quindi il polinomio traslato è: $z^5 - 2z^4 - 2z^3 - 8z^2 - 263z + 510$

Se applichiamo la formula (2) a questo polinomio otteniamo la stima $r \cong 8$. Quindi l'equazione del cerchio delle radici diventa

$$(x + 4.4)^2 + y^2 = 8^2$$

Il grafico mostra un netto miglioramento nella riduzione rispetto al caso precedente

Il cerchio esterno, di equazione

$$x^2 + y^2 = 44^2$$

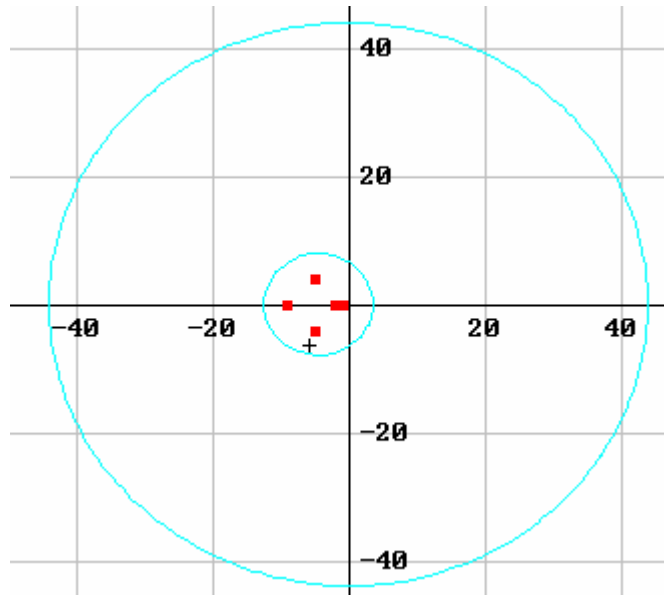
è stato ottenuto con l'aiuto della formula (1) applicata al polinomio originale

Il cerchio interno, di equazione

$$(x + 4.4)^2 + y^2 = 8^2$$

è stato ottenuto con la variante del polinomio centrato

Il miglioramento ottenuto compensa abbondantemente il lavoro effettuato per la traslazione del polinomio

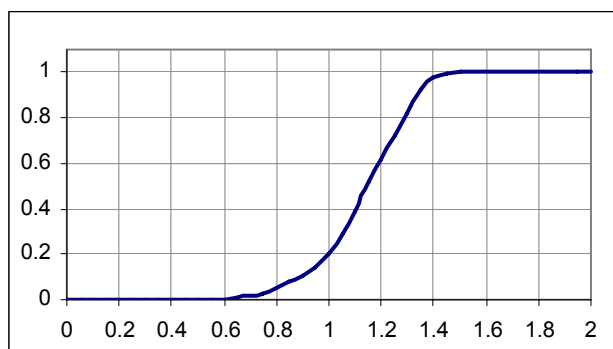


Un approccio statistico permette di restringere ulteriormente il cerchio scegliendo come coefficiente della formula (1) un valore $1.3 \div 1.5$

Infatti se consideriamo l'insieme dei polinomio centrati di grado n , il valore r fornito dalla (1) è una variabile casuale così come il raggio esatto delle radici ρ^* .

definendo la variabile casuale normalizzata $t = 2 \rho^*/\rho$ si può ottenere sperimentalmente la sua distribuzione statistiche cumulativa (F)

Polin. 10° grado
 Campioni = 790
 Media = 1.13
 Dev.pop = 0.17
 Max = 1.47
 Min = 0.63



Come si nota la variabile t è distribuita intorno al valore esatto ρ^* con una certa probabilità. Per valori $t > 1.4$ la probabilità di contenere tutte le radici è maggiore del 95%. Quindi

$$\rho^* = t \rho / 2 \Rightarrow \rho^* \cong 1.4 \rho / 2 \Rightarrow \rho^* = 1.4 \max \{ |a_{n-k}|^{1/k} \} \text{ per } k = 1, 2, \dots, n$$

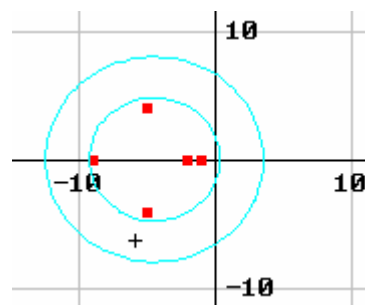
Che è la formula statistica per la stima del raggio delle radici dei polinomi centrati

Per il polinomio centrato precedente abbiamo

$$\rho^* \cong 1.4 \cdot 4 = 5.6$$

Il cerchio interno ha equazione:

$$(x + 4.4)^2 + y^2 = 5.6^2$$



Metodo iterativo delle potenze

Si tratta del metodo di Bernoulli, modificato per ottenere le radici reali o complesse di massimo e di minimo modulo di un'equazione polinomiale.

Data l'equazione

$$a_n z^n + a_{n-1} z^{n-1} + a_{n-2} z^{n-2} + a_2 z^2 + a_1 z + a_0 = 0$$

Si costruisce l'equazione alle differenze

$$a_n s_n + a_{n-1} s_{n-1} + a_{n-2} s_{n-2} + a_2 s_2 + a_1 s_1 + a_0 s_0 = 0$$

da cui

$$s_n = -(a_{n-1} s_{n-1} + a_{n-2} s_{n-2} + \dots + a_2 s_2 + a_1 s_1 + a_0 s_0) / a_n, \quad n = n+1, n+2, \dots(1)$$

Il rapporto fra i valori della successione $\{s_n\}$ tendono alla radice dominante cioè a quella di massimo modulo, se la radice è reale. Cioè:

$$\rho \cong |s_n / s_{n-1}| = \quad \text{per } n \gg 1 \quad (2)$$

Se la radice è complessa la (2) non converge. In tal caso si ottiene il modulo della radice per mezzo degli ultimi 4 valori della successione con la seguente formula

$$\rho \cong ((s_{n-1})^2 - s_n s_{n-2}) / ((s_{n-2})^2 - s_{n-1} s_{n-3})^{-1/2} \quad \text{per } n \gg 1 \quad (3)$$

La radice di modulo minimo si ricava trasformando il polinomio reciproco mediante il cambio di variabile $z \rightarrow 1/z$. Con questo trucco la radice più piccola diventa la più grande e quindi si può applicare il metodo delle potenze alla successione

$$q_n = -(a_1 q_{n-1} + a_2 q_{n-2} + a_3 q_2 + a_{n-1} q_1 + a_n q_0) / a_0, \quad n = n+1, n+2, \dots(4)$$

Le formule (2) e (3) applicate alla successione $\{q_n\}$ danno i reciproci ρ^{-1} dei raggi minimi.

Vediamo come funziona questo metodo con dei casi pratici

Esempio: $x^5 + 22x^4 + 190x^3 + 800x^2 + 1369x + 738$

I valori d'innescio della successione possono essere scelti arbitrariamente. Di solito si sceglie la sequenza $[0, 0, 0, 1]$. I valori delle prime 3 iterazioni sono riportati nella tabella seguente

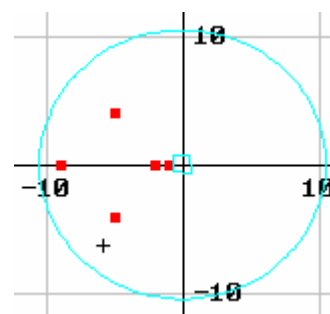
	Polinomio. reciproco		Polinomio originale	
n	q _n	ρ _{min}	s _n	ρ _{max}
5	-1.85501	0.539	-22	22
6	2.35706	0.787	294	13.36
7	-2.61898	0.900	-3088	10.50

Come si vede la successione $\{s_n\}$ tende a divergere rapidamente ma generalmente solo poche iterazioni sono sufficienti

In questo caso deduciamo il massimo raggio è 10.5 e che il minimo è 0.9. Quindi tutte le radici devono soddisfare la relazione

$$0.9 < |x| < 10.5.$$

Geometricamente l'area è costituita da due cerchi concentrici



Vediamo il caso di radice complessa dominante

$$x^5 + 18x^4 + 135x^3 + 450x^2 + 644x + 312 = 0$$

Le radici esatte dell'equazione sono $x = -6 - 4i$, $x = -6 + 4i$, $x = -3$, $x = -2$, $x = -1$

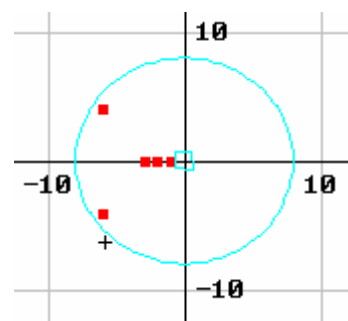
Il metodo delle potenze fornisce le seguenti iterazioni, dove per la sequenza $\{s_n\}$ abbiamo usato questa volta la formula (3).

n	Polinomio. reciproco		Polinomio originale	
	q_n	ρ_{\min}	s_n	ρ_{\max}
5	-2.06	-0.484	-18	
6	2.818	-0.732	189	11.61
7	-3.270	-0.861	-1422	8.660
8	3.525	-0.928	7537	7.682

In questo caso deduciamo il massimo raggio è 10.5 e che il minimo è 0.9 . Quindi tutte le radici devono soddisfare la relazione

$$0.9 < |x| < 8.$$

Geometricamente l'area è costituita da due cerchi concentrici



Metodo quoziente-differenza (QD)

L' algoritmo "quoziente-differenza", è una geniale generalizzazione del metodo delle potenze sviluppato da Rutishauser che ha il vantaggio di approssimare tutte le radici simultaneamente senza richiedere nessun valore d'innesco. Sotto condizioni abbastanza ragionevoli converge agli zeri reali fornendo anche una misura della precisione. Con una variante si riesce ad ottenere anche gli zeri complessi. L'ordine di convergenza è lineare per cui il suo uso principale è quello di fornire una approssimazione iniziale di tutti gli zeri del polinomio per il successivo passo di rifinitura con algoritmi più veloci, come ad esempio quello di Newton-Raphson o di Halley

I dettagli implementativi dell'algoritmo QD sono piuttosto complicati e ne esistono diverse varianti. Qui illustreremo i dettagli per l'implementazione su un foglio elettronico di calcolo.

Esso consiste in una tabella di righe e colonne: la riga i -esima rappresenta la corrispondente iterata e le colonne contengono alternativamente le differenze (d) e i quozienti (q)

Per semplicità di esposizione facciamo vedere lo sviluppo della tabella QD per il seguente polinomio

$$x^5 + 22x^4 + 190x^3 + 800x^2 + 1369x + 738$$

La tabella dovrà contenere in 11 colonne, alternativamente, $d_1, q_1, d_2, q_2, d_3, q_3, d_4, q_4, d_5, q_5, d_6$

Le colonne d_1 e d_6 sono ausiliare e contengono sempre 0. Poiché ad ogni riga corrisponde un passo d'iterazione conveniamo di chiamare $d^{(i)}, q^{(i)}$ i coefficienti del passo i -esimo. quindi, per esempio, i coefficienti al passo iniziale 0 saranno indicati come $d_1^{(0)}, q_1^{(0)}, d_2^{(0)}, q_2^{(0)}$, ecc. quelli al passo 1 saranno $d_1^{(1)}, q_1^{(1)}, d_2^{(1)}, q_2^{(1)}$, e così via.

Inizializzazione. La prima riga, che indicheremo con 0, serve ad innescare il processo iterativo e viene costruita con i coefficienti del polinomio nel modo seguente.

$$q_1^{(0)} = -a_4/a_5 \quad q_2^{(0)} = q_3^{(0)} = q_4^{(0)} = q_5^{(0)} = 0$$

$$d_1^{(0)} = 0, \quad d_2^{(0)} = a_3/a_4, \quad d_3^{(0)} = a_2/a_3, \quad d_4^{(0)} = a_1/a_2, \quad d_5^{(0)} = a_0/a_1, \quad d_6^{(0)} = 0$$

Da queste formule si deduce che il polinomio deve avere tutti i coefficienti non nulli. Questo è uno dei primo vincoli del algoritmo QD. Vedremo in seguito quale accorgimento si può adottare per ovviare a questo limite.

La prima riga della tabella apparirà simile alla seguente

n	d ₁	q ₁	d ₂	q ₂	d ₃	q ₃	d ₄	q ₄	d ₅	q ₅	d ₆
0	0	-22	8.6364	0	4.2105	0	1.7113	0	0.5391	0	0
1											

Le righe successive verranno generate usando le relazioni caratteristiche da cui l'algoritmo prende il nome.

$$q_k^{(i+1)} - q_k^{(i)} = d_{k+1}^{(i)} - d_k^{(i)} \tag{1}$$

$$d_k^{(i+1)} / d_k^{(i)} = q_k^{(i+1)} / q_{k-1}^{(i+1)} \tag{2}$$

dalla (1) si ricavano per primi i valori $q^{(i+1)}$ al passo $i+1$

$$q_k^{(i+1)} = q_k^{(i)} + d_{k+1}^{(i)} - d_k^{(i)} \tag{3}$$

con $k = 1, 2, \dots, 5$

dalla (2) si ricavano successivamente i valori $d^{(i+1)}$ al passo $i+1$

$$d_k^{(i+1)} = d_k^{(i)} \cdot (q_k^{(i+1)} / q_{k-1}^{(i+1)}) \tag{4}$$

con $k = 2, 3, \dots, 5$

I valori $d_1^{(i+1)} = d_6^{(i+1)}$, come già anticipato, sono sempre nulli.

Praticamente, per generare una nuova riga $(i+1)$, si alternano i seguenti due schemi

$d_k^{(i)}$	$q_k^{(i)}$	$d_{k+1}^{(i)}$
	$q_k^{(i+1)}$	

	$d_{k+1}^{(i)}$	
$q_k^{(i+1)}$	$d_{k+1}^{(i+1)}$	$q_{k+1}^{(i+1)}$

Per calcolare $q^{(i+1)}$ di una colonna si usano i tre valori adiacenti della riga precedente.

Successivamente per calcolare $d^{(i+1)}$ di una colonna si usano i valori adiacenti della stessa riga e il valore $d^{(i)}$ della riga precedente.

Le prime 8 iterazioni sono riportate nella tabella seguente.

n	d ₁	q ₁	d ₂	q ₂	d ₃	q ₃	d ₄	q ₄	d ₅	q ₅	d ₆
0	0	-22	8.6364	0	4.2105	0	1.7113	0	0.5391	0	0
1	0	-13.36	2.8602	-4.426	2.3777	-2.499	0.8026	-1.172	0.2479	-0.539	0
2	0	-10.5	1.3366	-4.908	1.9737	-4.074	0.3402	-1.727	0.113	-0.787	0
3	0	-9.167	0.6228	-4.271	2.6375	-5.708	0.1164	-1.954	0.052	-0.9	0
4	0	-8.544	0.1645	-2.257	9.6179	-8.229	0.0286	-2.018	0.0245	-0.952	0
5	0	-8.379	-0.141	7.1968	-23.81	-17.82	0.0032	-2.022	0.0119	-0.977	0
6	0	-8.521	-0.273	-16.47	8.669	5.9976	-0.001	-2.014	0.0058	-0.988	0
7	0	-8.794	-0.234	-7.532	3.0758	-2.673	-8E-04	-2.007	0.0029	-0.994	0

8	0	-9.028	-0.109	-4.223	4.1878	-5.749	-3E-04	-2.003	0.0014	-0.997	0
---	---	--------	--------	--------	--------	--------	--------	--------	--------	--------	---

Già da una prima ispezione vediamo che le colonne q_1, q_4, q_5 convergono alle radici reali mentre q_2, q_3 non convergono e quindi sospettiamo la presenza di due radici complesse coniugate.

Il metodo generale per estrarre le radici dalla tabella QD è il seguente

Ricerca radici singole reali - si calcola l'errore stimato della radice con la formula

$$e_i = 10(|d_{i,k}| + |d_{i+1,k}|) \tag{5}$$

se $(e_i / |q_i|) < \epsilon_r$ si accetta la radice i -esima q_i

Ricerca coppie radici complesse - si calcolano i valori delle successioni $\{s_i\}$ e $\{p_i\}$

$$s_i = q_{i,k} + q_{i,k+1} \tag{6}$$

$$p_i = q_{i-1,k} \cdot q_{i,k+1} \tag{7}$$

si calcola l'errore stimato con la formula

$$es_i = |s_i - s_{i-1}| \tag{8}$$

$$ep_i = |p_i - p_{i-1}|^{1/2} \tag{9}$$

se $(es_i / |s_i|) < \epsilon_r$ e $(ep_i / |p_i|) < \epsilon_r$

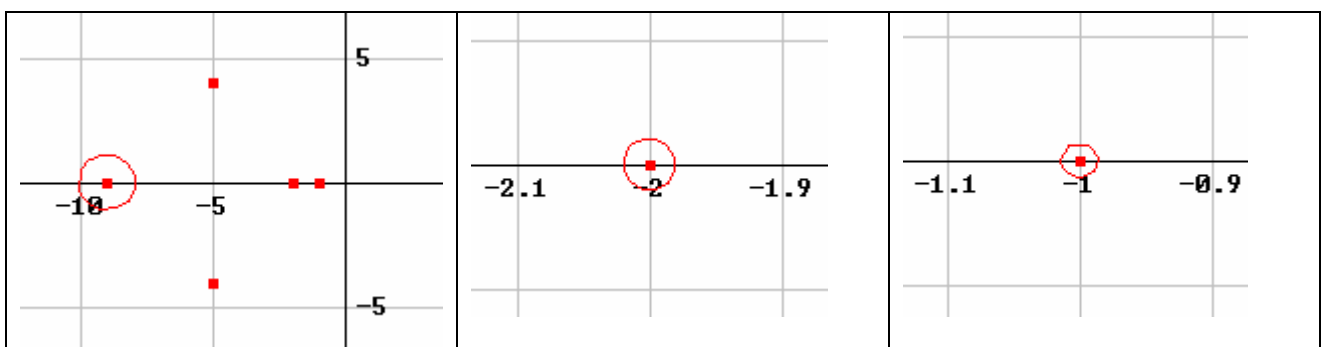
si eccettano i due valori $s = s_i$ e $p = p_i$

si trovano le radici soluzioni dell'equazione $x^2 + s \cdot x + p = 0$

Supponiamo di fermare l'iterazione alla riga 8. Gli errori percentuali secondo la stima (5) sono

err 1	err 2	err 3	err 4	err 5
12.12%	1017.7%	728.5%	0.86%	1.44%

Le aree per ogni singola radice reale sono mostrate nei seguenti grafici. Si noti la diverse scala della prima rispetto alle altre due.



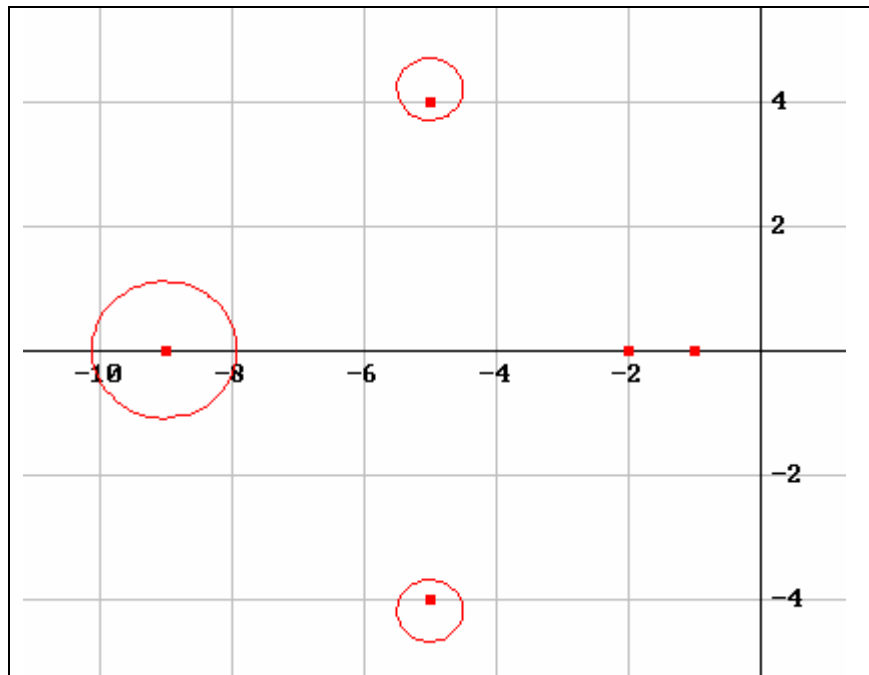
Vediamo ora le radici complesse. Dalle ultime 2 righe della tabella calcoliamo

$$s = q_2^{(8)} + q_3^{(8)} \cong -10 \quad , \quad p = q_2^{(7)} q_3^{(8)} \cong 43$$

Da cui si ricava la seguente equazione $(x^2 - 10x + 43) = 0$, che ha le radici $x_{1,2} \cong 5 \pm 4.2 i$
 La media delle stime fornite da (8) e (9) è di circa $er \cong 0.076$, per un errore assoluto di circa $er \cong 0.07 (5^2 + 4.2^2)^{1/2} \cong 0.5$. Possiamo prendere questa stima come raggio d'incertezza della radice e scrivere l'equazione dei cerchi della radici

$$(x + 5)^2 + (y - 4.2)^2 = 0.5^2 \quad e \quad (x + 5)^2 + (y + 4.2)^2 = 0.5^2$$

Da cui abbiamo il definitivo grafico dell'area delle radici ottenuto con le prime 8 iterazioni del algoritmo QD.



Caso di coefficienti nulli

Come abbiamo visto l'algoritmo QD non può essere innescato se uno o più coefficienti sono nulli. In tal caso si effettua una traslazione del polinomio con la sostituzione di variabile $x = z - a$ dove a è un valore arbitrario, (in genere si sceglie un valore random). Questa trasformazione elimina i valori nulli dei coefficienti e quindi si può applicare QD per ricercare le radici z_k del polinomio traslato. Per ottenere le radici del polinomio originale basta effettuare la trasformazione $x_k = z_k - a$

Ad esempio il polinomio $x^5 - 8x^2 - 10x - 1$ può essere trasformato in uno completo per mezzo della trasformazione di variabile $x = z + 1$ ottenendo

$$(z + 1)^5 - 8(z + 1)^2 - 10(z + 1) - 1 = z^5 + 5z^4 + 10z^3 + 2z^2 - 21z - 18$$

Ovviamente anche le radici sono tutte traslate di 1

Caso di polinomi con tutte radici reali distinte

Consideriamo il polinomio di Wilkinson di 6° grado

$$P(x) = x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120$$

Esso, come noto, ha le radici $x = 1, 2, 3, 4, 5$

Vediamo come funziona l'algorithmo QD per questo tipo di polinomi
 La tabella QD delle prime 12 iterazioni è:

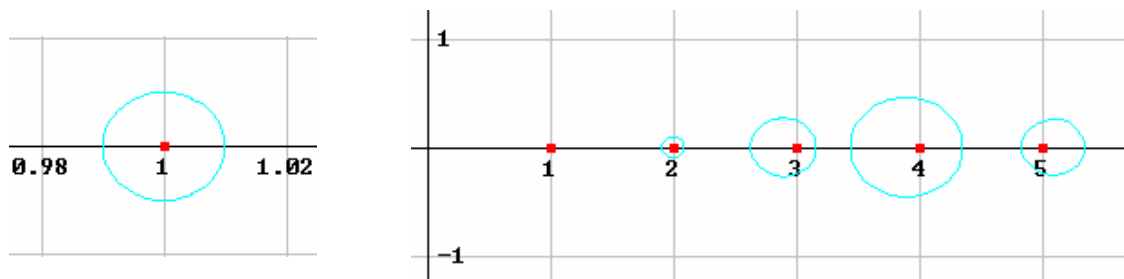
n	d ₁	q ₁	d ₂	q ₂	d ₃	q ₃	d ₄	q ₄	d ₅	q ₅	d ₆
0	0	15	-5.667	0	-2.647	0	-1.218	0	-0.438	0	0
1	0	9.333	-1.833	3.019	-1.253	1.4293	-0.664	0.7798	-0.246	0.438	0
2	0	7.5	-0.88	3.6	-0.702	2.0178	-0.395	1.1983	-0.14	0.6839	0
3	0	6.62	-0.502	3.7777	-0.432	2.3255	-0.246	1.4525	-0.08	0.8243	0
4	0	6.1178	-0.316	3.8476	-0.282	2.5114	-0.159	1.6193	-0.044	0.904	0
5	0	5.802	-0.211	3.8813	-0.192	2.6346	-0.105	1.7337	-0.024	0.9484	0
6	0	5.5907	-0.147	3.901	-0.134	2.7216	-0.07	1.8139	-0.013	0.9728	0
7	0	5.4433	-0.106	3.9148	-0.095	2.7855	-0.047	1.8706	-0.007	0.9858	0
8	0	5.3373	-0.078	3.9257	-0.069	2.8338	-0.032	1.9105	-0.004	0.9927	0
9	0	5.2593	-0.058	3.9351	-0.05	2.8709	-0.021	1.9385	-0.002	0.9963	0
10	0	5.201	-0.044	3.9433	-0.037	2.8997	-0.014	1.9579	-9E-04	0.9981	0
11	0	5.1567	-0.034	3.9507	-0.027	2.9221	-0.01	1.9714	-5E-04	0.999	0
12	0	5.1228	-0.026	3.9574	-0.02	2.9397	-0.007	1.9806	-2E-04	0.9995	0

Come si nota tutte le colonne "d" tendono a zero mentre le radici si leggono direttamente nelle colonne "q". E' sicuramente il caso più lampante in cui l'algorithmo rivela tutta la sua potenza approssimando simultaneamente tutte le radici.

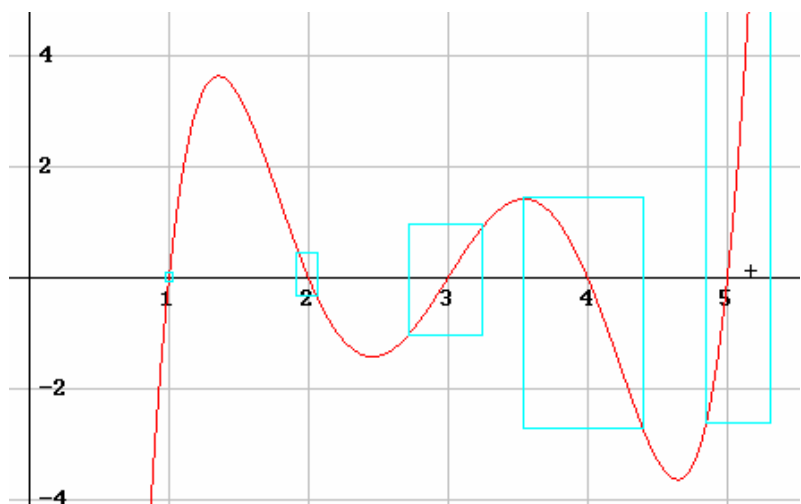
Calcolando l'errore per mezzo della (5) otteniamo le seguenti stime cautelative

X ₁	X ₂	X ₃	X ₄	X ₅
5.13	3.96	2.94	1.98	1.00
5.1%	11.7%	9.1%	3.4%	0.24%

Riportando in un grafico le consuete "bolle" si ha



Nella figura a destra è stato tracciato il grafico del polinomio nell'intorno degli zeri, insieme agli intervalli di separazione delle radici ottenuti. Tali intervalli servono per innescare gli algoritmi di raffinamento delle radici. Se si usano algoritmi a due punti si prendono gli estremi dell'intervallo. Se si usano algoritmi ad un punto si prende il valore centrale. In tutti i casi la convergenza è sicura



Costo computazionale. Chiaramente la restrizione dell'area delle radici con l'algoritmo QD da ottimi risultati, ma ovviamente ad un costo molto maggiore degli altri metodi.

Se il grado del polinomio è n allora per ogni iterazione vengono effettuate $2(2n-1)$ op.

NB. Nel conteggio non vengono inseriti i calcoli di predisposizione della riga 0.

Ad esempio il costo della tabella QD precedente è stato di $[2(2 \cdot 5 - 1)] \cdot 12 = 216$ op

Ricerca delle radici multiple

Abbiamo visto che la presenza di radici multiple degrada fortemente l'efficienza e la precisione di quasi tutti i metodi di approssimazione delle radici. Questo vale anche per i polinomi. Tuttavia per essi esistono delle tecniche per ridurre tale degrado.

Uno dei metodi più popolari è quello di sostituire al polinomio che contiene radici multiple un altro polinomio con le stesse radici ma singole. Questo problema può essere risolto teoricamente con la ricerca del massimo comun divisore dei polinomi (MCD o GCD).

Massimo comun divisore

Qui illustreremo il metodo per mezzo di esempi pratici

Dato il polinomio.

$$P(x) = x^5 + 12x^4 + 50x^3 + 88x^2 + 69x + 20$$

che ha le radici: $x_1 = -1$ con molteplicità $m = 3$, $x_2 = -4$, $x_3 = -5$ cioè: $P(x) = (x+1)^3(x+4)(x+5)$

Prendiamo la sua derivata

$$P'(x) = 5x^4 + 48x^3 + 150x^2 + 176x + 69$$

Poniamo: $P_0(x) = P(x)$ e $P_1(x) = P'(x)$. Facciamo la divisione dei due polinomi

$$P_0(x) / P_1(x) \Rightarrow Q_1(x), R_1(x)$$

Di questa divisione in realtà c'interessa solo il resto. Se il resto è un polinomio di grado > 0 , dividiamolo per il coefficiente della potenza massima per renderlo monico¹, e poniamo

$$P_2(x) = (-25/76) \cdot R_1(x) = x^3 + (120/19)x^2 + (183/19)x + (82/19)$$

Facciamo ancora la divisione dei due polinomi $P_1(x) / P_2(x)$

$$P_3(x) = (-361/675) \cdot R_2(x) = x^2 + 2x + 1$$

e poi la divisione dei due polinomi $P_2(x) / P_3(x)$

$$P_4(x) = 0$$

L'ultimo polinomio è una costante quindi l'algoritmo MCD si ferma. Poiché la costante è 0 significa che il polinomio P_3 è il MCD che divide esattamente sia il polinomio dato che la sua derivata.

Quindi le radici di P_3 , dovendo essere comuni tanto al polinomio originale $P(x)$ che alla sua derivata, devono essere necessariamente tutte e sole le radici multiple di $P(x)$.

Infatti

$$P_3(x) = x^2 + 2x + 1 = (x+1)^2$$

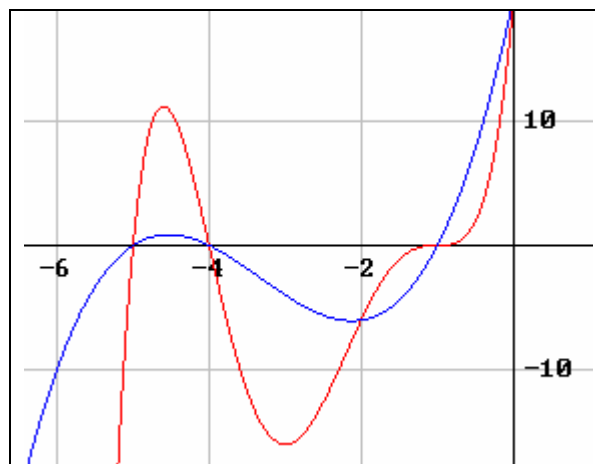
Osserviamo che la molteplicità è inferiore di un unità a quella del polinomio originale. Questo è un risultato generale: se la radice ha molteplicità "m" in $P(x)$, allora avrà molteplicità "m-1" nel polinomio MCD.

Dividiamo ora il polinomio originale per il MCD.

$$C(x) = P(x) / P_3(x) = x^3 + 10x^2 + 29x + 20$$

¹ Questo accorgimento non è indispensabile, ma servirà in seguito per ridurre gli errori di arrotondamento

Il polinomio ridotto $C(x)$, chiamato anche *cofattore* di P , ha solo radici singole e contiene tutte e solo le radici di $P(x)$ ed quindi è il polinomio cercato. I grafici seguenti mostrano il polinomio originale P (rosso) e quello ridotto C (blu)



Negli intorno delle radici la funzione $C(x)$ ha sempre pendenza diversa da zero per cui applicando gli algoritmi rootfinding a $C(x)$ si otterrà una migliore precisione proprio per le radici multiple. Per quelle singole invece non c'è un vero e proprio miglioramento: anzi, nei casi di arrotondamento decimale, la precisione può addirittura peggiorare.

Nel nostro esempio il polinomio MCD può essere risolto immediatamente. Con la radice trovata si può effettuare la deflazione intera del polinomio originale per tre volte di seguito. Il polinomio originale viene pertanto ridotto a

$$Pr(x) = x^2 + 9x + 20$$

Che può essere risolto agevolmente.

Massimo comun divisore con decimali

Il metodo sviluppato per arrivare al MCD è quello di Euclide, che per la sua eleganza concettuale e la buona efficienza è tuttora molto usato. Il suo costo computazionale è di circa $2n(n-1)$

Purtroppo solo per i polinomi di modesto grado può essere svolto il calcolo nella forma frazionale esatta. Nei casi pratici l'algoritmo MCD viene effettuato con i decimali e con precisione finita. La comparsa degli errori d'arrotondamento degrada la precisione con cui può essere trovato il polinomio cofattore $C(x)$.

Esempio. Il seguente polinomio

$$x^6 + 46x^5 + 878x^4 + 8904x^3 + 50617x^2 + 152978x + 192080$$

ha le radici: $x_1 = -7$, $m = 4$; $x_2 = -8$; $x_3 = -10$. Cioè: $P(x) = (x + 7)^4 \cdot (x + 8) \cdot (x + 10)$

Troviamo il polinomio ridotto con il metodo del MCD.

Sinteticamente i coefficienti dei polinomi sono riportati nella tabella seguente

	x^0	x^1	x^2	x^3	x^4	x^5	x^6
p0	192080	152978	50617	8904	878	46	1
p1	152978	101234	26712	3512	230	6	
p2	2775.1818	1532.364	316.90909	29.090909	1		
p3	343	147	21	1			
p4	4.186E-09	-2.4E-10	-1.01E-10				

I calcoli sono stati effettuati per mezzo di un foglio di calcolo con precisione standard 15 cifre significative. Notiamo che l'ultima riga mostra un valore di circa 1E-9 anziché essere zero, come prevede la teoria. Questo fenomeno, classico del calcolo numerico, è dovuto agli errori di arrotondamento. Le celle, anche quelle visualizzate come intere, se osservate attentamente rivelano la presenza dei decimali.

Infatti i coefficienti del polinomio MCD e di quello ridotto ottenuti mediante il calcolo in virgola mobile sono.

	a0	a1	a2	a3
MCD	343.0000000049060	147.0000000017460	21.0000000001279	1.0000000000000
Ridotto	559.9999999913070	205.9999999977430	24.9999999998721	1.0000000000000

E' evidente qui che il polinomio MCD esatto dovrebbe essere $x^3+21x^2+147x+343$, ma questo esempio mostra che nei casi reali gli arrotondamenti possono essere molti insidiosi e, se non controllati, vanificare anche gli algoritmi più brillanti.

Ad esempio, appare chiaro che la semplice rivelazione della costante zero per interrompere l'algoritmo non può funzionare. Si deve adottare un altro approccio.

Definiamo qui la norma del polinomio $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$

$$\| P(x) \| = (a_0^2 + a_1^2 + \dots + a_{n-1}^2 + a_n^2)^{1/2} \tag{1}$$

Il criterio di arresto dell'algoritmo MCD dovrà verificare la seguente relazione

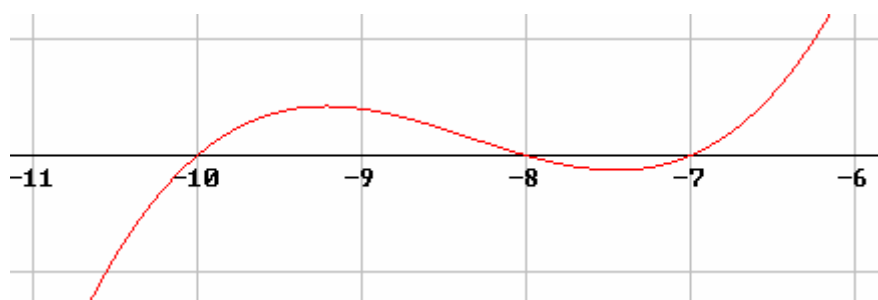
$$\| P_k(x) \| < \epsilon \| P_{k-1}(x) \|$$

Dove ϵ è la precisione fissata. Se la relazione è vera l'algoritmo s'interromperà alla k-esima iterazione. Nel nostro esempio il fermo si è avuto alla 4° iterazione perchè abbiamo posto

$$\epsilon = 1E-10, \| P_3(x) \| \cong 1000, \| P_4(x) \| \cong 4.2E-9.$$

Vediamo ora quale precisione possiamo aspettarci nella ricerca delle radici. Calcoliamo ora tutte le singole radici del polinomio ridotto, approssimato, $C(x)$ con uno dei metodi rootfinder precedentemente visti. Ad esempio il classico Newton-Rapshon

Dal grafico di $C(x)$ desumiamo i possibile valori d'innescio: $x_0 = -11, -8.5, -6.5$

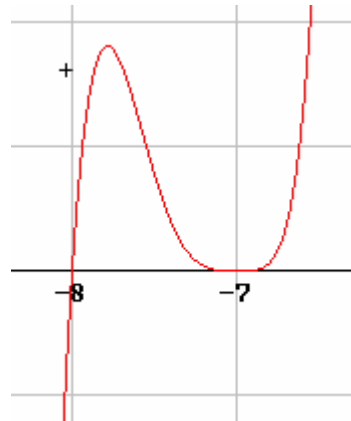
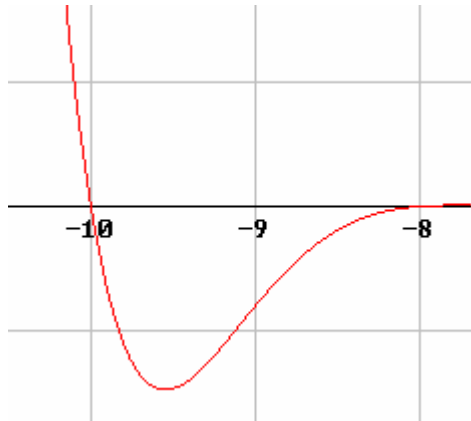


Otteniamo i seguenti valori

x	err
-7.0000000001279	2.8E-10
-7.9999999998721	5.9E-10
-10.0000000001817	1.82E-10

Come si vede tutte le radici sono state trovate con una buona e uniforme precisione, che dell'ordine di grandezza di quella dei coefficienti del polinomio ridotto. Soprattutto è da rilevare la precisione della quadrupla radice $x = -7$

Cosa avremmo ottenuto invece se la stessa ricerca fosse stata effettuata con il polinomio originale? Dal grafico di $P(x)$ desumiamo i possibile valori d'innescio: $x_0 = -10.2, -8.5, -6.8$

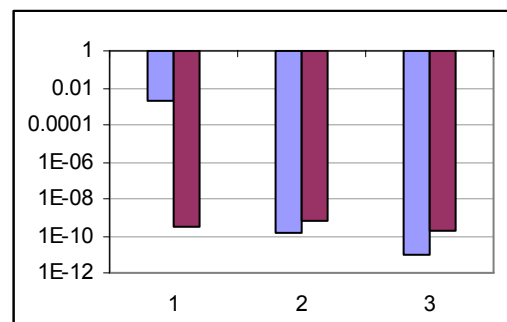


Otteniamo i seguenti valori

x	err
-7.00223808742748	0.002238
-8.00000000015983	1.6E-10
-9.99999999999033	9.67E-12

In questo caso vediamo una grande differenza di precisione: si va da 1E-11 fino a 1E-3 della radice quadrupla.

Il grafico a fianco confronta la precisione delle tre radici con il metodo MCD (rosso) e il metodo diretto (celeste). La terza radice è di un ordine di grandezza superiore nel metodo diretto ma la radice multipla ha una precisione bassa. Il metodo MCD ha livellato la precisione ad un valore alto ed uniforme. Chiaramente, in questo caso, il costo del processo è stato ben ripagato.



Esempio. Il seguente polinomio

$$x^9 - 10x^8 + 42x^7 - 100x^6 + 161x^5 - 202x^4 + 200x^3 - 144x^2 + 80x - 32$$

ha le radici: $x_1 = 2, m = 5; x_2 = \pm i, m = 2$. Cioè: $P(x) = (x - 2)^5 \cdot (x^2 + 1)^2$

Troviamo il polinomio ridotto con il metodo del MCD.

Sinteticamente i coefficienti dei polinomi sono riportati nella tabella seguente

	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9
p0	-32	80	-144	200	-202	161	-100	42	-10	1
p1	80	-288	600	-808	805	-600	294	-80	9	
p2	40.72727	-65.4545	69.81818	-61.8182	23.63636	4.636364	-5.45455	1		
p3	16	-32	40	-40	25	-8	1			
p4	7.74E-13	-1.3E-12	1.49E-12	-1.4E-12	7.32E-13	-1.4E-13				

L'arresto si è avuto alla 4° iterazione perchè: $\varepsilon = 1E-12$, $\| P_3(x) \| \cong 6016$, $\| P_4(x) \| \cong 2.6E-12$.

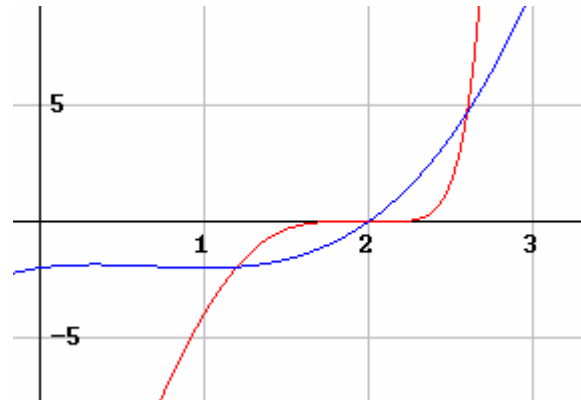
I coefficienti del polinomio MCD e di quello ridotto ottenuti mediante il calcolo in virgola mobile sono.

	a_0	a_1	a_2	a_3	a_4	a_5	a_6
GCD	16	-32	40	-40	25	-8	1
Ridotto	-2	1	-2	1			

La figura riporta il grafico del polinomio ridotto (blu) e di quello originale (rosso) Approssimiamo l'unica radice reale usando prima il polinomio originale e poi quello ridotto.

Come punto d'innescio scegliamo $x_0 = 2.5$, buono per entrambe le funzioni

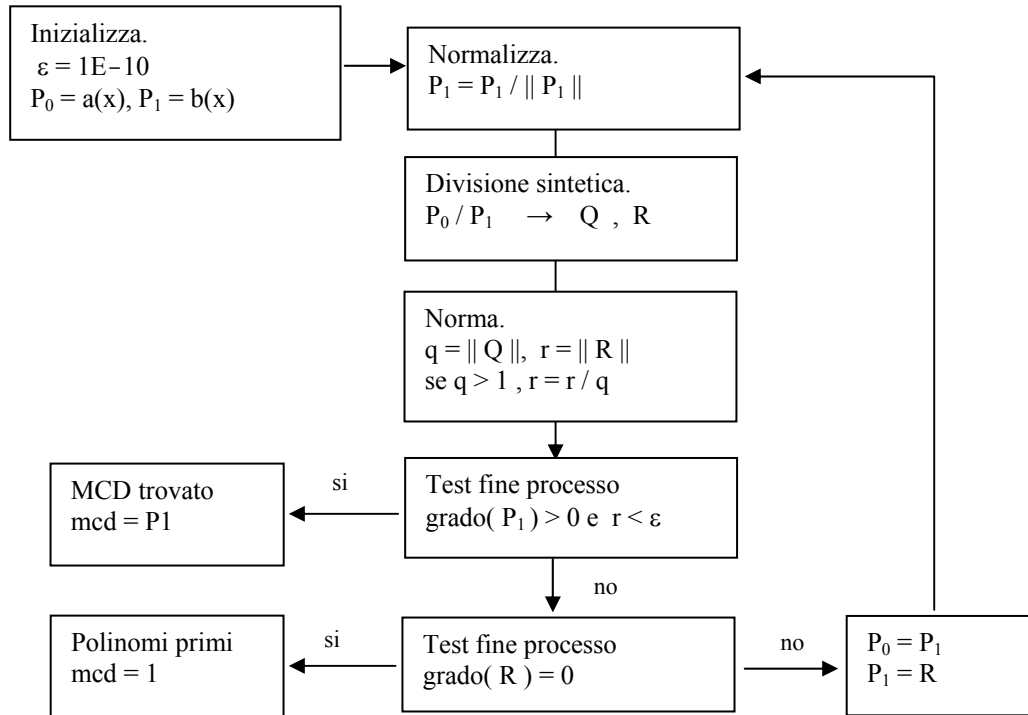
	iter.	x	err
diretto	28	2.0012696	1.27E-03
GCD	6	2	1.00E-15



Per entrambi i casi è stato usato il metodo Newton-Raphson. Si noti che la variante per le radici multiple del metodo di Newton non avrebbe portato miglioramento perchè agisce solo sulla velocità di convergenza ma non sulla precisione finale. Per sfuggire alla "barriera" della molteplicità bisogna, come in questo caso, cambiare polinomio.

MCD - Algoritmo pratico di Euclide.

Per calcolare il MCD di due polinomi $a(x)$ e $b(x)$, con $\text{grado}(a) \geq \text{grado}(b)$, al fine di ridurre la propagazione degli errori e arrestare il processo di Euclide, è conveniente adottare il seguente algoritmo pratico per aritmetica in precisione standard (15 cifre)



Ad esempio per i seguenti polinomi la successione dei polinomi $R(x)$ è riportate nella tabella

a(x)	b(x)
31680	-97488
-97488	227176
113588	-191469
-63823	74336
18584	-13450
-2690	888
148	7
1	

x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7
31680	-97488	113588	-63823	18584	-2690	148	1
-97488	227176	-191469	74336	-13450	888	7	
-94.5125	223.064	-191.106	75.63561	-14.081	1		
48.4334	-88.652	51.2438	-12.0271	1			
-16	24	-9	1				
-9.93E-13	1.27E-12	-2.78E-13					

Per motivi di spazio sono stati riportati solo alcuni decimali, ma i coefficienti del MCD calcolati dall'algoritmo sono riportati per esteso nella colonna "coeff" della tabella seguente

coeff	coeff (arr.)
-15.99999999994830	-16
23.99999999993140	24
-8.99999999998298	-9
1.000000000000000	1

In trucco molto semplice per migliorare la precisione finale è quella di arrotondare all'intero tutti i coefficienti (coeff. arr.). Questo può essere fatto ogni volta che il polinomio $a(x)$ è intero e monico. Si osserva che il polinomio $b(x)$ è, in questo caso, la derivata di $a(x)$.

MCD - Test dei residui dei cofattori

L'algoritmo delle divisioni successive del paragrafo consente di trovare praticamente l'MCD fra due polinomi di grado simile, ma può fallire se i gradi sono molto diversi. In tal caso l'algoritmo può ritornare un MCD diverso da 1 anche quando i polinomi sono primi relativi.

Esempio. calcoliamo l' MCD fra il polinomio a(x) del paragrafo precedente e $b(x) = x^2 - 937 + 950$

x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7
31680	-97488	113588	-63823	18584	-2690	148	1
950	-937	1					
-1.0149735	1						
-2.2204E-16							

Il resto finale è molto basso, dell'ordine di 2E-16, e quindi la penultima riga dovrebbe contenere i coefficienti del polinomio MCD. Il risultato è però errato poichè i due polinomi sono in realtà primi fra loro e quindi il loro MCD deve essere 1.

La insufficiente selettività del metodo di Euclide è nota e nel corso del tempo sono stati sviluppati varianti e metodi per superare questo problema. Un metodo abbastanza semplice e relativamente poco costoso per verificare il MCD ottenuto sfrutta i residui dei cofattori.

Essendo:

$$a(x) = \text{mcd} \cdot Ca + Ra \quad , \quad b(x) = \text{mcd} \cdot Cb + Rb$$

Se l'MCD è esatto i resti Ra e Rb devono essere nulli o comunque con coefficienti molto piccoli. Praticamente, dato il polinomio mcd approssimato si effettua la divisione sintetica per ottenere i cofattori Ca e Cb e i loro resti Ra e Rb, da cui si calcola la norma dei resti normalizzata:

$$r_a = \| Ra \| / \| Ca \| \text{ e } r_b = \| Rb \| / \| Cb \|$$

Se il grado n_a del polinomio a(x) è simile al grado n_b di b(x) allora il peso dei residui r_a e r_b è equivalente e $r \cong \text{media}(r_a, r_b)$; viceversa, se i gradi sono molto differenti allora l'errore ponderato medio può essere stimato meglio con la formula:

$$r \cong (r_a \cdot n_a^2 + r_b \cdot n_b^2) / (n_a^2 + n_b^2)$$

In base al valore di r si distinguono tre casi

$r < 10^{-9}$	Test positivo; MCD accettato
$r > 10^{-7}$	Test negativo; MCD scartato
$10^{-9} \leq r \leq 10^{-7}$	Test dubbio; necessaria multiprecisione

Nel caso precedente si ottengono i seguenti valori che danno come indice di errore $r \cong 2.1E-5$ e quindi il test scarta, giustamente, il gcd trovato

	a	b
$\ R\ $	1.96519	1.016E-12
$\ C\ $	88115.4	935.98
r	2.23E-05	1.086E-15
n	7	2

MCD - La matrice di Sylvester

Il polinomio MCD può essere ottenuto anche attraverso la cosiddetta matrice di Sylvester

Vediamo subito un esempio pratico

Dati i polinomi $a(x)$ e $b(x)$ di cui si vuole calcolare il loro MCD

$$a(x) = x^5 - 13x^4 + 63x^3 - 139x^2 + 136x - 48$$

$$b(x) = 5x^4 - 52x^3 + 189x^2 - 278x + 136$$

I gradi dei polinomi sono $n_a = 5$ e $n_b = 4$. Si costruisce la seguente matrice quadrata S_n di dimensione $n = n_a + n_b = 9$.

5	-52	189	-278	136	0	0	0	0
1	-13	63	-139	136	-48	0	0	0
0	5	-52	189	-278	136	0	0	0
0	1	-13	63	-139	136	-48	0	0
0	0	5	-52	189	-278	136	0	0
0	0	1	-13	63	-139	136	-48	0
0	0	0	5	-52	189	-278	136	0
0	0	0	1	-13	63	-139	136	-48
0	0	0	0	5	-52	189	-278	136

Nella prima riga, a partire dalla prima colonna, si scrivono i coefficienti del polinomio di grado inferiore $b(x)$ mentre nella seconda riga quelli del polinomio di grado superiore $a(x)$. Le righe seguenti sono ottenute nel solito modo spostandosi ogni volta di una colonna fino al riempimento della matrice (9 x 9). Si procede quindi alla riduzione triangolare inferiore della matrice, ad esempio, con il metodo di Gauss.

1	-10.4	37.8	-55.6	27.2	0	0	0	0
0	1	-10.4	37.8	-55.6	27.2	0	0	0
0	0	1	-10.4	37.8	-55.6	27.2	0	0
0	0	0	1	-10.4	37.8	-55.6	27.2	0
0	0	0	0	1	-7.764	17.82	-11.06	0
0	0	0	0	0	1	-7.58	16.9	-10.32
0	0	0	0	0	0	1	-5	4
0	0	0	0	0	0	0	1E-14	5E-14
0	0	0	0	0	0	0	-1E-14	1E-14

Come si vede, la norma delle ultime due righe è molto piccola pertanto vengono considerate numericamente nulle. Di conseguenza, anche il determinante è nullo e il rango della matrice è $r = 7$. Quando il determinante della matrice di Sylvester è nullo i polinomi $a(x)$ e $b(x)$ hanno un divisore comune diverso da 1. Al contrario, se il determinante è diverso da zero allora i polinomi sono primi.

Il metodo fornisce anche i coefficienti del MCD nell'ultima riga non nulla della matrice ridotta. La precisione è comparabile con quella del metodo delle divisioni di Euclide e la selettività è sensibilmente migliore. Per contro il costo è maggiore e diventa rilevante per polinomi di alto grado. Una variante molto interessante ed efficiente, ma anche molto complessa, di questo metodo è stata sviluppata per ottenere una accurata struttura delle radici multiple di polinomi di alto grado¹

¹ "MultRoot - A Matlab package computing polynomial roots and multiplicities", Zhonggang Zeng, Northeastern Illinois University, 2003

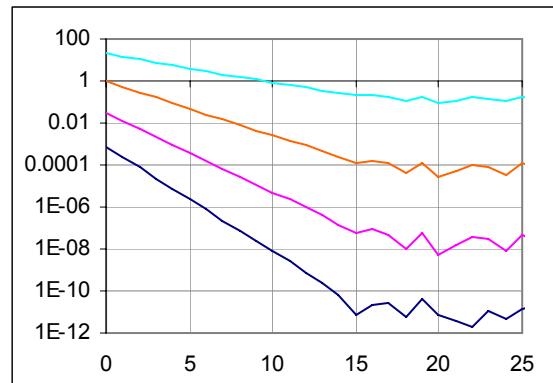
Metodo delle derivate

Un metodo meno conosciuto ma molto accurato è quello che riduce la molteplicità attraverso il calcolo delle derivate. Concettualmente è molto semplice: se la radice $x = a$ di un polinomio $P(x)$ ha molteplicità "m" allora tutte le sue $m-1$ derivate sono nulle nello stesso punto. Cioè si verifica:

$$P(a) = 0, P^{(1)}(a) = 0, P^{(2)}(a) = 0 \dots P^{(m-1)}(a) = 0 \quad (1)$$

Si osserva che, l'operazione di derivazione di un polinomio a coefficienti interi non introduce errori e quindi il metodo sembra molto interessante. Tuttavia le relazioni precedenti non possono essere utilizzate direttamente perchè sono contemporaneamente valide solo nel punto esatto della radice.

In generale possiamo disporre solo di un'approssimazione della radice $x = a \pm \epsilon$. Come abbiamo visto l'effetto barriera non permette a nessun algoritmo di ridurre oltre un certo limite il valore dell'errore ϵ . Il grafico mostra le curve del polinomio e delle sue derivate durante una tipica iterazione verso una radice di molteplicità 4. Si osserva che dopo il 15° passo tutte le curve smettono di scendere e si stabilizzano a valori molto superiori allo zero.



La curva $P(x)$ e la più bassa e in ordine crescente le derivate P', P'', P''' . Come si nota, aspettarsi che vengano verificate esattamente le relazioni (1) è una pura utopia.

Per sfruttare le relazioni (1), nei casi reali dobbiamo ricorrere ad un approccio numerico differente. Il metodo, chiamato anche "drill-down" si articola per fasi successive. Supponiamo di aver ricavato un'approssimazione della radice per mezzo di un algoritmo qualsiasi¹. Si definiscono le seguenti quantità

$$\begin{aligned} r_0 &= \text{miglior approssimazione della radice ottenuta con il polinomio } P(z) \\ D_0 &= |P(r_0)|, \text{ residuo del polinomio } P(z) \text{ in } r_0 \\ E_0 &= E_0 = |r_0 - x_0|, \text{ stima dell'errore iniziale} \end{aligned}$$

Vogliamo vedere attraverso un test se la radice trovata ha molteplicità $m > 1$. Per questo applichiamo il metodo iterativo alla derivata prima $P^{(1)}(x)$ e definiamo in generale:

$$\begin{aligned} r_k &= \text{miglior approssimazione della radice ottenuta con il polinomio } P^{(k)}(z) \\ D_k &= |P^{(k)}(r_k)|, \text{ residuo del polinomio } P^{(k)}(z) \text{ in } r_k \\ E_k &= |r_k - r_{k-1}|, \text{ stima dell'errore} \end{aligned}$$

Test per molteplicità $m = 2$ ($k = 1$): se $[(E_1 \leq 10 \cdot E_0) \text{ e } (D_1 \leq 10 \cdot D_0)] \Rightarrow$ radice r_1 confermata con molteplicità $m = 2$; in caso contrario si prende la radice r_0 con molteplicità $m = 1$.

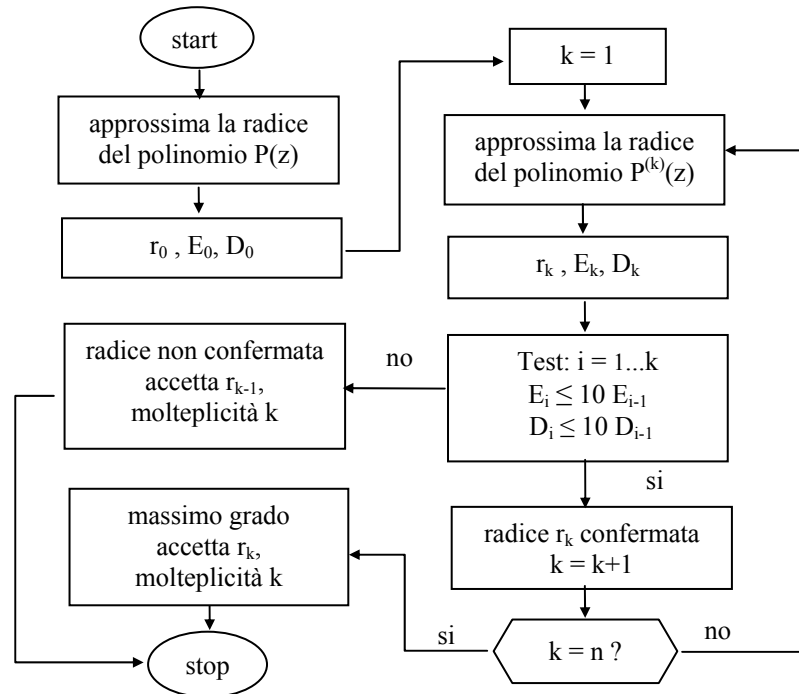
Se la molteplicità 2 è confermata si applica di nuovo il test alla derivata seconda $P^{(2)}(x)$ e così via. In generale il test per molteplicità $m = k+1$ è:

$$\text{se } [(E_i \leq 10 \cdot E_{i-1}) \cap (10 \cdot D_i \leq D_{i-1}), i = 1 \dots k], \Rightarrow \text{ radice } r_k \text{ confermata con molteplicità } m = k+1. \text{ In caso contrario radice } r_{k-1} \text{ confermata con molteplicità } m = k$$

¹ Vedremo in seguito che un algoritmo che si adatta bene a questo metodo è quello di Newton. Con questa variante viene chiamato metodo di "Newton generalizzato".

Nota. Il coefficiente 10 viene inserito empiricamente per tenere conto delle irregolarità random che si verificano in prossimità del limite di macchina ("bumping").

Il seguente schema di flusso spiega passo per passo la procedura di raffinamento e test di una radice multipla.



Si osserva che il metodo del drill-down non necessita di ipotesi iniziale circa la molteplicità della radice. La radice multipla viene confermata o rigettata solo a valle del test. Quindi, in teoria, il test dovrebbe essere ripetuto per ogni nuova radice trovata.

Il metodo procede per "prove ed errori", ed è quindi intrinsecamente lento. Occorre mettere in evidenza però che è molto importante evitare l'errata attribuzione di una molteplicità perchè essa si riflette in modo disastroso in tutto il processo successivo di deflazione. Il polinomio ridotto sarebbe completamente sbagliato. E meglio rigettare un molteplicità dubbia ed accettare una radice con molteplicità inferiore perchè il danno risulta più limitato. Sotto questo aspetto il metodo è molto cautelativo.

Il costo del metodo è indubbiamente alto in quanto l'intero processo di approssimazione viene ripetuto almeno 2 volte per le radici singole, 3 volte per radice doppie, ecc. Per polinomi di alto grado il tempo di esecuzione può incidere notevolmente a sfavore.

Per contro questo metodo si è rivelato contemporaneamente robusto nel riconoscimento della molteplicità ed accurato nell'approssimazione della radice multipla, senza degradare in alcun modo la precisione delle radici singole.

Stima della molteplicità

Abbiamo visto che i criteri di test della molteplicità sono sempre piuttosto costosi e quindi servirebbe avere un modo, più semplice possibile, per decidere quando è il caso di applicarli veramente. Il metodo seguente si basa sui valori della successione $\{x_n\}$ prodotta da un algoritmo per l'approssimazione delle radice. Per "x" sufficientemente vicino alla radice "r" di molteplicità "m" possiamo scrivere:

$$P(x) \cong k(x - r)^m$$

$$P'(x) \cong k \cdot m(x - r)^{m-1}$$

Da cui, dividendo la prima per la seconda si ha

$$P(x) / P'(x) \cong (x - r) / m$$

$$\Rightarrow m \cdot \Delta x_n \cong r - x_n \tag{1}$$

Dove $\Delta x_n = x_{n+1} - x_n = -P(x_n) / P'(x_n)$.

La (1) può essere applicata a due punti consecutivi della successione

$$m \cdot \Delta x_n \cong r - x_n$$

$$m \cdot \Delta x_{n+1} \cong r - x_{n+1}$$

Da cui, eliminando r , si ha il valore stimato di m

$$m \cong -(x_{n+1} - x_n) / (\Delta x_{n+1} - \Delta x_n) = \Delta x_n / (\Delta x_n - \Delta x_{n+1}) \tag{2}$$

Osserviamo che usando l'algoritmo di Newton gli incrementi Δx_n sono già calcolati per cui il costo aggiuntivo della (2) è solo 2 op per iterazione.

La formula (2) dà dei valori attendibili in prossimità della radice e sempre prima del limite inferiore di macchina (breakdown). Inoltre può essere facilmente ingannata dalle radici raggruppate (cluster roots), per cui non dovrebbe essere usata da sola. Deve essere usata invece come indicatore della presenza o meno di una radice multipla, da verificare e confermare per mezzo di test più robusti

Ad esempio consideriamo il polinomio:

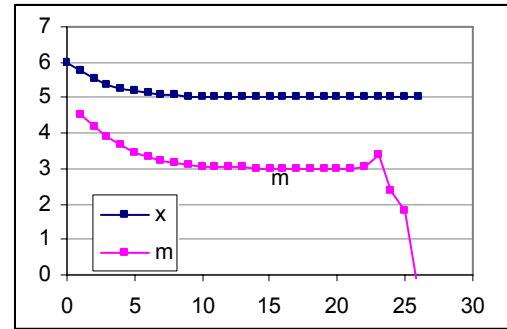
$$-1250 + 3000x - 3375x^2 + 2155x^3 - 818x^4 + 180x^5 - 21x^6 + x^7$$

Supponiamo di avere isolato una radice reale in un intervallo [4, 6]

Usiamo l'algoritmo di Newton per l'approssimazione della radice partendo da $x_0 = 6$.

n	x	P	P'	Δx	m	P
0	6	442	1704	-0.259		442
1	5.7406	142.96	708.16	-0.202	4.5097	142.96
2	5.5387	45.672	297.43	-0.154	4.1783	45.672
3	5.3852	14.404	126.26	-0.114	3.8901	14.404
4	5.2711	4.4856	54.137	-0.083	3.6532	4.4856
5	5.1882	1.3808	23.419	-0.059	3.4678	1.3808
6	5.1293	0.4209	10.204	-0.041	3.3286	0.4209
7	5.088	0.1273	4.4717	-0.028	3.2274	0.1273
8	5.0596	0.0383	1.9679	-0.019	3.1556	0.0383
9	5.0401	0.0115	0.8687	-0.013	3.1056	0.0115
10	5.0269	0.0034	0.3843	-0.009	3.0713	0.0034

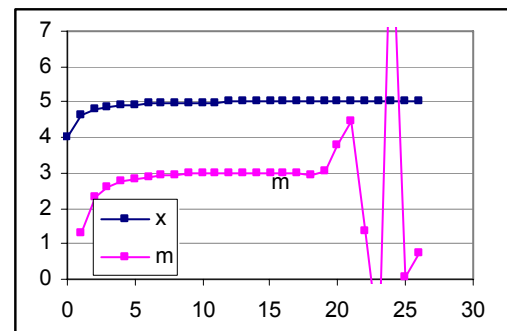
Nel grafico a destra sono riportati le successioni di valori di x_n e m . Dopo il transitorio iniziale, si vede chiaramente la tendenza a stabilizzarsi sui valori $x = 5$ e $m = 3$. La molteplicità stimata è quindi di 3. Si osserva che i valori finali di m sono del tutto inattendibili. Questo significa che l' algoritmo ha raggiunto la barriera inferiore oltre il quale non può andare. In questa situazione l' algoritmo si ferma e i valori delle ultime iterazioni ($n > 20$) di "m" devono essere scartati.



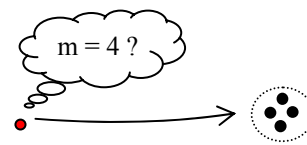
Se, ad esempio, l' algoritmo di ricerca ha impostato un soglia di $m > 1.5$, dopo i primi 5 - 10 passi può attivare la routine di test della molteplicità (drill-down test) e confermare o meno la molteplicità. In tal modo si riesce ad aumentare sensibilmente l' efficienza dell' intero processo.

Il grafico a destra mostra le successioni per un valore d' innesco di $x_0 = 4$. Anche in questo caso la molteplicità viene rivelata in modo piuttosto evidente già dopo poche iterazioni del transitorio iniziale.

Anche qui si notano le oscillazioni aleatorie di "m" negli ultimi passi (saturazione inferiore).

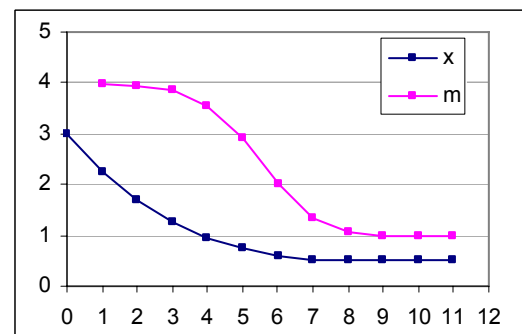


La formula di stima della molteplicità è molto semplice ed ha un costo molto basso, Tuttavia può anche essere facilmente ingannata. Questo accade, ad esempio, quando l' algoritmo si avvicina ad un gruppo di radici vicine (cluster)



A grande distanza, l' algoritmo "vede" il cluster come se fosse una radice unica con molteplicità data dalla somma delle singole molteplicità; se ad esempio ci sono 4 radici singole, la molteplicità "apparente" risulta 4. Man mano che la distanza relativa si riduce, la molteplicità stimata si riduce fino a portarsi al valore unitario in prossimità di una della 4 radici.

Questo fenomeno può essere facilmente messo in evidenza. Ad esempio il polinomio $P(x) = x^4 - 1/16$ ha 4 radici di raggio 0.5 intorno all' origine. Inneschiamo il metodo di Newton da $x_0 = 3$ e tracciamo la successione dei valori "x" e di "m". Come è mostrato chiaramente, nelle prime iterazioni il valore di m è di circa 4 e discende progressivamente a 1 man mano che x si avvicina alla radice $r = 0.5$. Per $x \cong 0.52$ (+4%), $m \cong 1.33$ (+33%)



In questi casi la stima fornirebbe un "falso allarme", ma questo come abbiamo detto non è un problema in quanto la conferma della molteplicità viene svolta da altri metodi più sofisticati.

Ricerca delle radici intere

Dato un polinomio a coefficienti interi, le sue eventuali radici intere possono essere trovate con un processo molto semplice ed estratte per mezzo della deflazione intera. Il polinomio ridotto che ne deriva è esente da errori di arrotondamenti per cui anche le eventuali radici non intere che verranno successivamente estratte potranno guadagnare sensibilmente in accuratezza.

Il metodo si basa sul seguente asserto: se un polinomio ha una radice intera allora questa deve essere anche un divisore esatto del termine noto a_0 .

Possiamo quindi con un numero finito di tentativi determinare tutte le radici intere del polinomio. Vediamo con un esempio

$$P(x) = 16x^6 - 58x^5 - 169x^4 + 326x^3 - 230x^2 + 384x - 45$$

L'insieme dei divisori esatti di 45 è $Z_{45} = \{\pm 1, \pm 3, \pm 5, \pm 9, \pm 15, \pm 45\}$. I divisori interi vanno presi con il segno sia positivo che negativo¹. Se il polinomio ha una radice intera essa deve appartenere a questo insieme.

Calcoliamo successivamente il polinomio per ogni numero $x_i \in Z_{45}$: $P(x_i)$ per $i = 1, 2 \dots 12$.

Il calcolo viene eseguito praticamente con il metodo di Ruffini-Horner che dà anche il polinomio ridotto in caso di test positivo. Partendo dai numeri più bassi, alternando i segni, troviamo la prima radice intera $x_1 = -3$. Infatti si ha

-3	16	-58	-169	326	-230	384	-45
		-48	318	-447	363	-399	45
	16	-106	149	-121	133	-15	0

Il coefficienti del polinomio ridotto si trovano nella terza riga in basso. Riapplichiamo il processo a questo polinomio ripartendo dalla radice appena trovata e proseguendo alternativamente fino a trovare la seconda radice intera $x_2 = 5$. Infatti si ha

5	0	16	-106	149	-121	133	-15
		0	80	-130	95	-130	15
	0	16	-26	19	-26	3	0

Proseguendo non troviamo più alcuna radice; ne consegue che le uniche radici intere sono $x_1 = -3$, $x_2 = 5$, per cui il polinomio dato può essere fattorizzato nel modo seguente

$$P(x) = P_1(x) (x + 3) (x - 5) = (16x^4 - 26x^3 + 19x^2 - 26x + 3) (x + 3) (x - 5)$$

Il polinomio residuo P_1 può essere ora sottoposto al normale processo di ricerca delle radici. I vantaggi sono un grado minore e i coefficienti esatti per cui le successive radici non saranno affette dagli errori della deflazione delle due radici intere

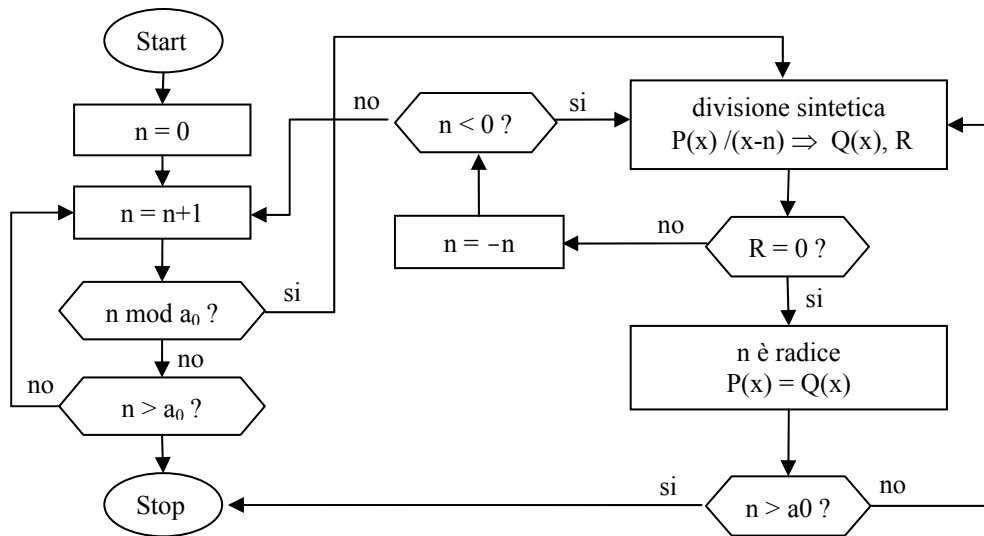
Dal punto di vista pratico il metodo può essere realizzato con due algoritmi differenti: per tentativi successivi (brute force attack) o per intervalli (intervals attack)

Tentativi successivi

L'algoritmo è simile a quello della ricerca dei numeri primi. Si provano in successione tutti i numeri "n" compresi far 1 e a_0 , alternando i segni positivi e negativi .

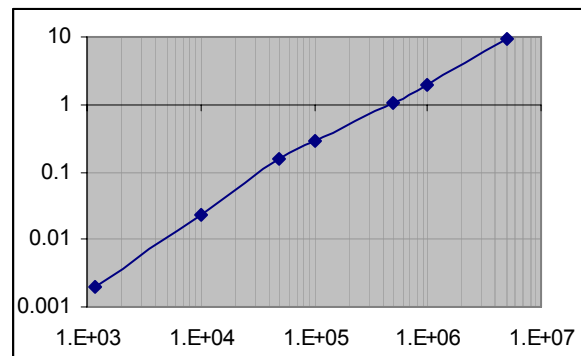
¹ Un ulteriore riduzione dell'insieme può essere fatta se il polinomio ha tutti coefficienti positivi. In tal caso si dimostra che tutte le radici devono essere nel semipiano sinistro e di conseguenza si scartano tutti i divisori positivi.

Se $(n \bmod a_0) = 0$ allora si effettua la divisione sintetica $P(x)/(x-n)$. In caso di resto nullo ($R = 0$) si sostituisce il polinomio originale con il quoziente e si riparte dal numero "n".



Il tempo di ricerca di questo metodo cresce approssimativamente in modo lineare con il valore assoluto $|z_1|$ della più piccola radice intera del polinomio. La tabella seguente ed il suo grafico mostrano l'andamento del tempo in funzione di $|z_1|$.

$ z_1 $	time (sec)
1200	0.002
10000	0.023
50000	0.156
100000	0.289
500000	1.031
1000000	1.953
5000000	9.273



Come si vede, il metodo ha delle risposte accettabili per $|z_1| < 10000$. Per valori superiori i tempi di ricerca diventano molto lunghi. Si osserva che ne caso che il polinomio non abbia radici intere il tempo di ricerca cresce con il valore assoluto del termine noto $|a_0|$.

Il costo computazionale dell'algorithmo è in generale molto alto e viene giustificato solo per il guadagno di accuratezza globale. Il computo dipende da molti fattori quali il grado del polinomio, il numero delle radici intere, la loro distribuzione, ecc. Qui possiamo fare una stima del costo per trovare la radice intera minima $|z_m|$, con $1 \leq |z_m| \leq |a_0|$.

Il numero dei divisori "n_d" di un numero intero "m" in generale possono essere stimati con la formula $n_d \approx 10 \log_{10}(m/16)$. Per ogni numero viene effettuato il test modulare (1 op); per ogni divisore trovato vengono effettuate 2 divisioni sintetiche (3n+2 op); una col segno positivo e una col quello negativo. Quindi il costo totale è di circa $[|z_m| + 20 \cdot (3n+2) \cdot \log_{10}(|z_m|/16)]$ op.

Per un polinomio di 10° grado e $|z_m| = 1000$ il costo è di circa 1610; se $|z_m| = 1E6$ il costo sale proporzionalmente a circa 1E6.

Nel caso che il polinomio no abbia radici intere il costo può essere calcolato ponendo $|z_m| = |a_0|$

Tentativi per intervalli

Quando le radici intere sono molto grandi oppure per alti gradi del polinomio, il tempo di ricerca del metodo diretto può diventare inaccettabile.

Per ovviare a questo inconveniente la ricerca viene ristretta ad opportuni intervalli contenenti le radici intere. Come abbiamo visto uno dei metodi più efficaci per l'isolamento delle radici reali è il metodo QD.

Prendiamo ad esempio il polinomio

$$x^5 - 1840x^4 + 1064861x^3 - 224912240x^2 + 64952800x - 1371280000$$

prendiamo le prime 4 iterazioni dell'algorithmo QD

n	d ₁	q ₁	d ₂	q ₂	d ₃	q ₃	d ₄	q ₄	d ₅	q ₅	d ₆
0	0	1840	-578.7	0	-211.2	0	-0.289	0	-211.1	0	0
1	0	1261.3	-168.6	367.52	-121.2	210.92	0.2887	-210.8	211.41	211.12	0
2	0	1092.6	-64.04	414.93	-97.12	332.43	0.0003	0.2893	-211.3	-0.289	0
3	0	1028.6	-23.77	381.85	-109.2	429.55	-1E-04	-211	211.32	211.03	0
4	0	1004.8	-7.012	296.38	-198.6	538.8	-7E-08	0.2891	-211.3	-0.289	0

Osserviamo che la colonna q₁ converge alla radice reale x = 1004.8 (±70)

Questo significa che la radice (l'algorithmo QD non ci dice se intera o meno) si trova in un intervallo 934 < x < 1075.

Partendo da n = 934, con solo 66 tentativi, l'algorithmo di ricerca sequenziale delle radici intere trova la radice x = 1000.

	1	-1840	1064861	-224912240	64952800	-1371280000
1000	0	1000	-840000	224861000	-51240000	1371280000
	1	-840	224861	-51240	13712800	0

Inoltre, come sottoprodotto abbiamo anche i coefficienti del polinomio ridotto

Il costo stimato della ricerca si trova con il seguente ragionamento:

$$C_{1000} = [| 1000| + 20 (3 \cdot 5 + 2) \cdot \text{Log}_{10}(| 1000| / 16)] \cong 1610 \text{ op}$$

$$C_{934} = [| 934| + 20 \cdot (3 \cdot 5 + 2) \cdot \text{Log}_{10}(| 934| / 16)]. \cong 1534 \text{ op}$$

Quindi il costo per la ricerca 934 < x < 1000 è di circa: C₁₀₀₀ - C₉₃₄ = 76 op

A cui si deve aggiungere il costo di 4 iterazioni dell'algorithmo QD: 4(4n-2) = 72 op

In totale la ricerca della radice intera è costata circa 150 op che è 10 volte inferiore alla ricerca sequenziale da 1 a 1000 (C₁₀₀₀ ≅ 1610)

Si fa notare che l'algorithmo di ricerca delle radici intere è intrinsecamente esatto, cioè non introduce alcun errore di approssimazione. Pertanto per le radici intere è indifferente partire dal valore minimo o dal massimo.

Algoritmi

La ricerca degli zeri dei polinomi si differenzia dalla quella degli zeri delle funzioni non lineari soprattutto per il dominio che in generale è il piano complesso. Purtroppo in questo dominio non è possibile applicare i concetti di ricerca per intervalli analoghi a quelli sviluppati per le radici reali. Per questo dominio è possibile applicare quasi tutti i metodi ad un punto utilizzati per le generiche funzioni non lineari: molto usati sono i metodi Newton-Raphson e Halley in aritmetica complessa. Ovviamente il costo del calcolo è molto maggiore di quello standard. Possiamo stimare il costo computazionale di un operazione complessa¹ circa 4 volte quello di un operazione in virgola mobile standard.

Un altro aspetto molto importante che diversifica i metodi per polinomio riguarda l'utilizzo delle derivate, che nel caso dei polinomi può essere sempre effettuato per qualunque ordine e grado. In aggiunta, per coefficienti interi, l'operazione di derivazione è virtualmente esente da errori. Quindi il ricorso alle derivate è ampiamente usato.

Metodo Newton-Raphson complesso

E' il metodo più popolare anche per la ricerca degli zeri, reali e complessi dei polinomi. Viene largamente usato sia nelle routine di calcolo automatico sia nel calcolo manuale

$$z_{i+1} = z_i - P(z_i) / P'(z_i)$$

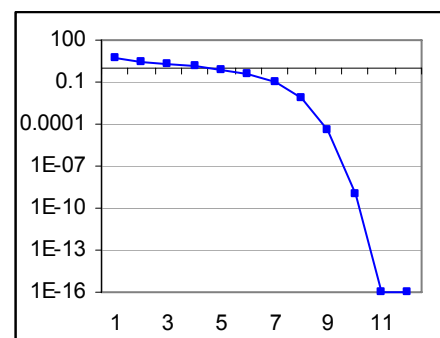
Come abbiamo già visto, è un metodo del 2° ordine. Per i polinomi viene usato quasi sempre con l'aritmetica complessa. Il suo costo computazionale è di $4(4n-1)$ / Iterazione.

E' relativamente semplice da implementare con uno foglio di calcolo se si adottano speciali "addin" per il calcolo numerico complesso.²

Il problema tipico di questo metodo è una certa sensibilità al punto d'innesco: questo significa che la velocità media di convergenza dipende sensibilmente dal punto d'innesco. In questi casi, viene raccomandato di scegliere un punto "sufficientemente vicino" alla radice ma, come abbiamo già mostrato, questo concetto è un po' vago e può portare a strane situazioni. Ad esempio si voglia trovare la radice reale del polinomio, partendo da $x_0 = -15$

$$x^3 + 4x^2 + 4x + 3$$

x	P(x)	P'(x)	errore
-15	-2532	559	4.529517
-10.470483	-748.24755	249.12918	3.0034521
-7.4670309	-220.17781	111.53341	1.9740974
-5.4929335	-64.016999	50.573487	1.2658213
-4.2271122	-17.966597	23.788534	0.7552629
-3.4718493	-4.521207	12.386418	0.3650133
-3.106836	-0.8061411	8.1026018	0.0994916
-3.0073444	-0.0516807	7.0736055	0.0073061
-3.0000382	-0.0002677	7.0003824	3.824E-05
-3	-7.311E-09	7	1.044E-09



Il grafico mostra la traiettoria dell'errore $|x_i - r|$ in funzione delle iterazioni. Tipicamente la forte riduzione dell'errore avviene nelle ultime 2÷3 iterazioni mentre il resto serve per avvicinarsi alla radice.

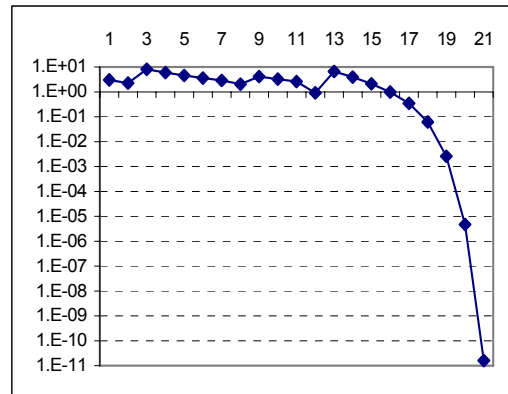
¹ Si ricorda che il nostro approccio è quello di mantenere le operazioni indifferenziate. Pertanto il costo di cui si parla è la media fra tutte le operazioni aritmetiche elementari

² I calcoli sono stati effettuati con MS Excel XP e l'addin Xnumbers.xla v.4.5 (Foxes Team)

Come si vede, l'algorithmo ha approssimato la radice $r = -3$ con un errore minore di $1E-16$, in 10 iterazioni anche se il valore iniziale è piuttosto distante dalla radice.

Vediamo invece cosa succede se partiamo invece da $x_0 = -0.001$.

x	P(x)	P'(x)	error
-0.00100000	2.996004E+00	3.992003E+00	3.00E+00
-0.75150144	1.828598E+00	-3.177483E-01	2.25E+00
5.00336264	2.484004E+02	1.191278E+02	8.00E+00
2.91820419	7.358766E+01	5.289338E+01	5.92E+00
1.52695902	2.199451E+01	2.321048E+01	4.53E+00
0.57934779	6.854421E+00	9.641714E+00	3.58E+00
-0.13156536	2.540699E+00	2.999405E+00	2.87E+00
-0.97863290	1.979099E+00	-9.558961E-01	2.02E+00
1.09177951	1.343643E+01	1.631018E+01	4.09E+00
0.26797332	4.378375E+00	6.359216E+00	3.27E+00
-0.42053535	1.950887E+00	1.166267E+00	2.58E+00
-2.09329696	2.981779E+00	3.993008E-01	9.07E-01
-9.56079851	-5.435495E+02	2.017402E+02	6.56E+00
-6.86649441	-1.596176E+02	9.051428E+01	3.87E+00
-5.10304238	-4.613654E+01	4.129879E+01	2.10E+00
-3.98590198	-1.271963E+01	1.977503E+01	9.86E-01
-3.34268535	-3.026206E+00	1.077915E+01	3.43E-01
-3.06193915	-4.529939E-01	7.630901E+00	6.19E-02
-3.00257604	-1.806551E-02	7.025780E+00	2.58E-03
-3.00000473	-3.309246E-05	7.000047E+00	4.73E-06
-3.00000000	-1.117457E-10	7.000000E+00	1.60E-11



Come si vede l'algorithmo spende la maggior parte del tempo nella ricerca caotica di un punto d'attacco.

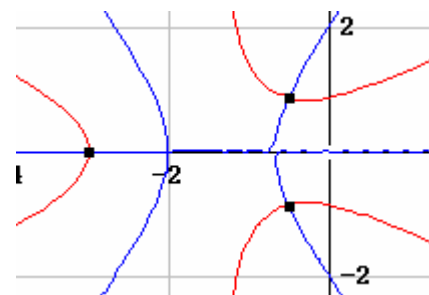
Questo esempio mostra che il punto d'innescio più vicino alla radice, può produrre degli inaspettati risultati disastrosi. Paradossalmente il punto d'innescio $x_0 = -100$, molto lontano dalla radice, converge in meno di 20 iterazioni senza problemi, mentre per $x_0 = 0$ la convergenza fallisce

Per convergere ad una radice complessa il metodo di Newton deve essere innescato con un valore complesso. Ad esempio, si vuole trovare le radici complesse del polinomio $x^3 + 4x^2 + 4x + 3$

Per prima cosa localizziamo le radici con il metodo grafico tracciando le curve definite implicitamente dalle seguenti equazioni:

$$x^3 + 4x^2 + x(4 - 3y^2) - 4y^2 + 3 = 0$$

$$y(3x^2 + 8x - y^2 + 4) = 0$$



Si vede che un punto d'innescio possibile è $z_0 = -1+i$

i	z		P(z)		P'(z)		dz	
0	-1	1	1	-2	-4	2	-0.4	0.3
1	-0.6	0.7	0.746	-0.147	-1.19	3.08	-0.12295345	-0.19470305
2	-0.47704655	0.89470305	-0.1628118	0.058920035	-1.5351328	4.596734385	0.022173388	0.028013954
3	-0.49921994	0.866689095	-0.00404623	0.002385261	-1.49954782	4.33750191	0.000779278	0.000663439
4	-0.49999921	0.866025656	-2.2697E-06	3.02105E-06	-1.49999739	4.330132359	7.85053E-07	2.52223E-07
5	-0.5	0.866025404	3.53495E-13	2.42584E-12	-1.5	4.330127019	4.7495E-13	-2.4616E-13
6	-0.5	0.866025404	-4.4409E-16	0	-1.5	4.330127019	3.17207E-17	9.15697E-17

In poche iterazioni viene trovata la radice complessa $z = -0.5 \pm i 0.866$

Facciamo vedere che il metodo di Newton può convergere anche ad una radice reale partendo da un punto d'innescio complesso come ad esempio $z = -4 + i$

i	z		P(z)		P'(z)		dz	
0	-4	1	-5	19	17	-16	-0.71376147	0.44587156
1	-3.28623853	0.55412844	-0.63781703	5.431081824	9.187007828	-6.49296187	-0.3249332	0.361521729
2	-2.96130533	0.192606711	0.444614531	1.267438413	6.506253135	-1.88135	0.011080673	0.198007211
3	-2.97238601	-0.0054005	0.189649754	-0.0363244	6.726060173	0.053110224	0.028151863	-0.00562284
4	-3.00053787	0.000222339	-0.00376629	0.00155757	7.005379421	-0.00222411	-0.0005377	0.000222168
5	-3.00000017	1.70709E-07	-1.199E-06	1.19496E-06	7.000001713	-1.7071E-06	-1.7128E-07	1.70709E-07
6	-3	4.17699E-14	0	2.92389E-13	7	-4.177E-13	-2.4925E-27	4.17699E-14
7	-3	0	0	0	7	0	0	0

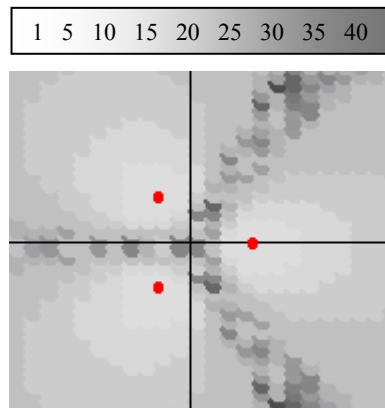
Al contrario, partendo da un valore iniziale reale l'algoritmo può convergere solo ad una radice reale (se esiste). Da questo si deduce che un punto d'innescio valido in generale deve essere necessariamente complesso.

Il punto d'innescio

La velocità di convergenza dell'algoritmo di Newton varia a seconda del punto innescio $P_0(x, y)$. Ci sono punti in cui la convergenza è estremamente rapida (6÷8 iterazioni) ed altri in cui la convergenza diventa molto lenta o addirittura fallisce. In passato sono stati intrapresi degli studi per analizzare le zone di convergenza dell'algoritmi iterativi in generale e di quello di Newton in particolare. Questi studi hanno portato, come noto, ai sorprendenti ed interessanti risultati della geometria dei frattali esplorati negli anni '80 con dei supercomputer in grado di restituire immagini bianche/nere ed analizzati da pionieri come Mandelbrot, Hubbard e molti altri. In questo contesto ci interessa solo visualizzare le zone di "buona" e "cattiva" convergenza dell'algoritmo di Newton.

Spettri di convergenza

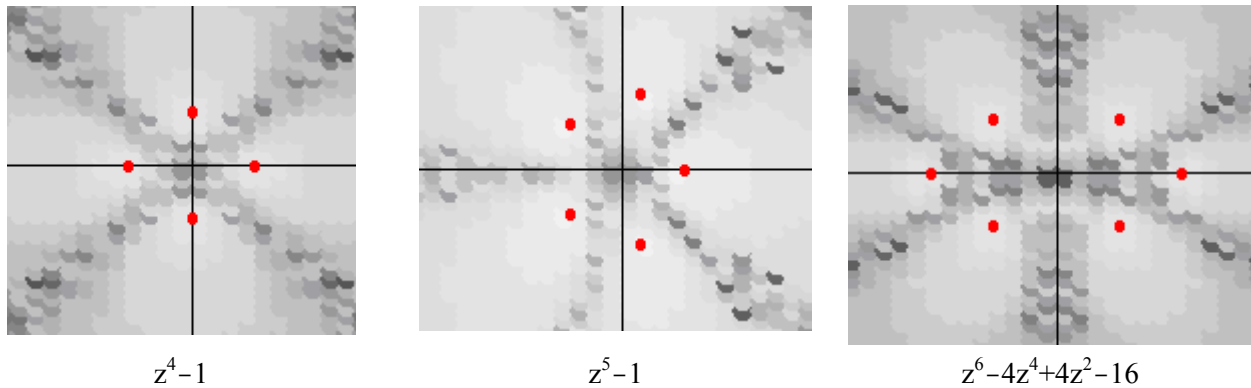
Per far questo scegliamo un semplice polinomio di test come ad esempio $z^3 - 1$. Applichiamo lo schema di Newton prendendo come valore d'innescio $z_0 = x + i y$, dove $P(x, y)$ è un punto generico del piano complesso. Misuriamo il numero d'iterazioni necessari per la convergenza, associando al numero una scala di grigi arbitraria. Di conseguenza ad un punto $P(x, y)$ viene a corrispondere un certo numero d'iterazioni e quindi una tonalità di grigio più o meno intenso a seconda della difficoltà di convergenza. Applicando tale iterazione ad ogni punto del piano si viene a creare una configurazione di zone chiare e scure



La peculiarità è che i confini non sono ben definiti ma presentano continuamente zone chiare e scure caoticamente mischiate. Le tre direttrici in cui si concentrano le zone caotiche sono le bisettrici del 1° e 4° quadrante e l'asse negativo delle x. Notiamo che la zona interna al cerchio delle radici, dove partono le tre direttrici, presenta un' alta "turbolenza". Dall'immagine dello spettro di convergenza si deduce chiaramente che innescare lo schema iterativo di Newton partendo da queste zone significa avere un'alta probabilità di fallimento o comunque una bassa efficienza. Al contrario, la probabilità di successo aumenta considerevolmente se si sceglie un punto molto più esterno al cerchio delle radici. L'analisi effettuata riguarda il polinomio centrato, ma le considerazioni si possono estendere a qualunque altro polinomio; basta traslare l'origine degli assi nel centro del polinomio stesso.

Viene spontaneo domandarsi se queste zone di "turbolenza" esistono anche per altri polinomi. La risposta è affermativa: non solo esistono ma il numero delle direttrici aumenta con il grado e la zona interna alle radici presenta delle configurazioni di chiaro/scuro sempre più caotica ed intricata

Nelle immagini seguenti sono rappresentate gli spettri di convergenza di tre differenti polinomi

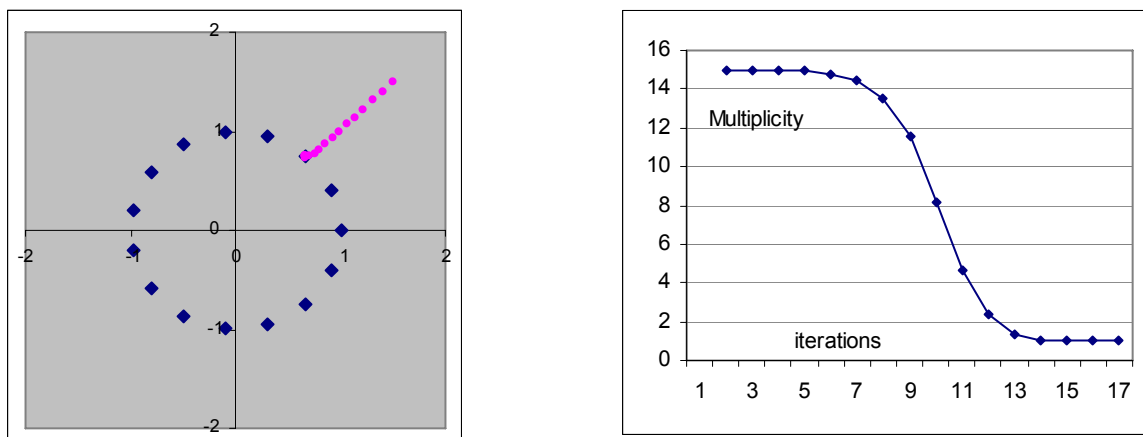


Per evitare la zona centrale delle radici, le routine automatiche che usano il metodo di Newton vengono fatte partire con un numero complesso random con modulo 2 ÷ 3 volte il raggio stimato delle radici. Conseguenza di questa tattica è che le radici con modulo più grande verranno incontrate ed estratte per prime e quindi è conveniente usare la deflazione reciproca per ridurre la propagazione dell'errore.

Molteplicità e clustering

Nelle pagine precedenti abbiamo mostrato che una formula di stima della molteplicità per l'algoritmo di Newton è data da $m \cong \Delta x_n / (\Delta x_n - \Delta x_{n+1})$ e abbiamo visto che si hanno valori attendibili solo in prossimità della radice; da lontano si hanno invece delle sovrastime, accentuate dalla presenza di radici singole vicine (clustering). Vediamo il seguente esempio.

Ricerca una radice del polinomio $z^{15} - 1$, con punto d'innesco $z_0 = (1.6, 1.6)$



Come si nota, l'algoritmo si avvicina inizialmente come se ci fosse una radice unica "virtuale" di molteplicità 15 situata nell'origine; solamente in prossimità della radice "vera" l'effetto delle altre radici scompare e la formula di stima dà valori corretti. L'effetto del clustering si manifesta anche nella velocità di convergenza: molto lenta nelle iterazioni di avvicinamento e veloce nelle ultime iterazioni.

Metodo Halley complesso

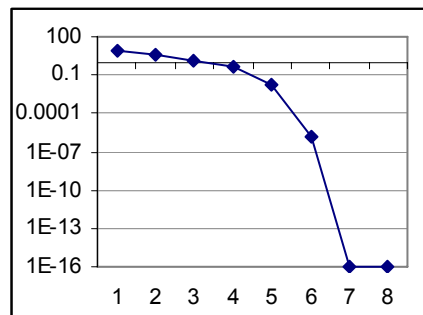
Questo efficiente metodo del 3° ordine usa la formula iterativa

$$x_{i+1} = x_i - P(x_i) \cdot P'(x_i) / ([P'(x_i)]^2 - 0.5 \cdot P(x_i) \cdot P''(x_i))$$

Applicato ai polinomi di grado n, il costo computazionale in aritmetica complessa è dato da: $(4[2n+2(n-1)+2(n-1)+7]) = 4(6n+1) / \text{Iterazione}$.

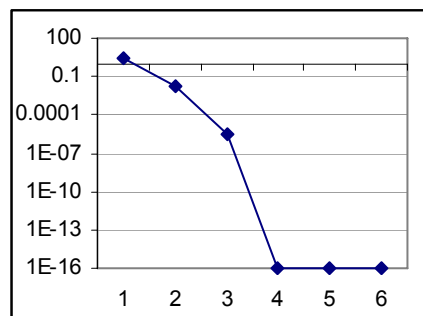
Vediamo come lavora in un caso pratico. Ad esempio si voglia trovare la radice reale del polinomio: $x^3 + 4x^2 + 4x + 3$, partendo da $x_0 = -15$.

i	x	P(x)	P'(x)	P''(x)	dx
0	-15	-2532	559	-82	6.782933737
1	-8.217066263	-314.605	140.8240	-41.30239	3.322527165
2	-4.894539098	-38.00819	36.71322	-21.36723	1.481640878
3	-3.41289822	-3.813105	11.6404	-12.47738	0.397330795
4	-3.015567425	-0.110187	7.156401	-10.09340	0.015566067
5	-3.000001358	-9.50E-06	7.000013	-10.00000	1.35797E-06
6	-3	0	7	-10	0



In soli 6 iterazioni l' algoritmo ha approssimato la radice con un errore inferiore a 1E-16, con un costo globale di 114 op. L'algoritmo di Newton nelle stesse condizioni ha impiegato 10 iterazioni, con un costo di 110 op. Quindi nonostante la cubica convergenza, il metodo di Halley non è risultato, in questo caso, il più efficiente. L'esperienza ha mostrato che in generale questo metodo, risulta sensibilmente più stabile. Vediamo infatti il comportamento nel punto critico $x_0 = -0.001$.

x	P(x)	P'(x)	P''(x)	dx
-0.001	2.996003999	3.992003	7.994	-3.0194082
-3.020408201	-0.14494838	7.20533149	-10.1224492	0.020405161
-3.00000304	-2.1282E-05	7.0000304	-10.0000182	3.04028E-06
-3	0	7	-10	0



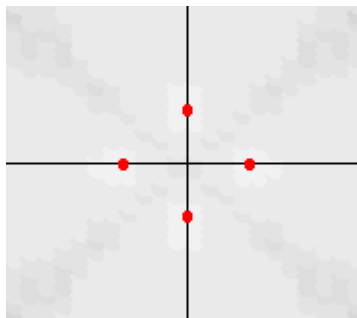
In soli 4 passi, il metodo iterativo di Halley ha raggiunto la radice con un errore inferiore a 1E-16

E' rimarchevole il fatto che in questa stessa situazione il metodo iterativo di Newton ha impiegato ben 20 iterazioni e fallito per $x_0 = 0$. Qui, invece per $x_0 = 0$ otteniamo la radice addirittura in un passo! Chiaramente le doti di stabilità del metodo di Halley sono superiori. La stabilità, insieme alla cubica convergenza, hanno reso molto interessante questo metodo per la ricerca degli zeri dei polinomi.

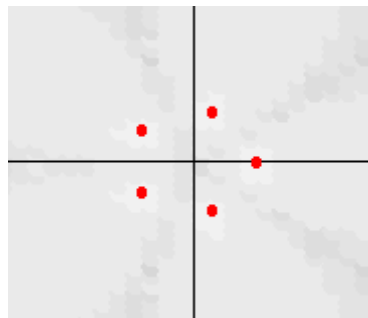
Il metodo converge anche alle radici complesse, se innescato con un numero complesso. Infatti se partiamo da $z_0 = -1+i$, otteniamo

z	P(z)	P'(z)	P''(z)	dz
-1	1	1	-2	-4
-0.458599	0.8407643	0.05539	0.21461	-1.15850
-0.500034	0.8660603	-9.97E-05	-0.0002	-1.50035
-0.5	0.8660254	2.11E-13	6.66E-14	-1.5
-0.5	0.8660254	-4.44E-16	0	-1.5

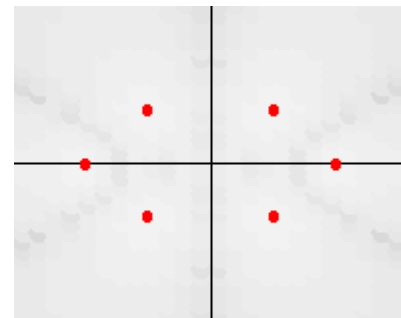
Nelle immagini seguenti sono rappresentate gli spettri di convergenza di tre differenti polinomi ottenuti con il metodo di Halley



$z^4 - 1$



$z^5 - 1$



$z^6 - 4z^4 + 4z^2 - 16$

Come si nota, le zone scure presenti nei diagrammi di Newton, sono qui molto ridotte: questo significa che l'algoritmo di Halley è molto più stabile dal punto di vista della convergenza.

Stima della molteplicità

Anche per l'algoritmo di Halley è possibile stimare la molteplicità della radice dai valori della successione per mezzo della formula:

$$m = (\Delta x_n + \Delta x_{n+1}) / (\Delta x_n - \Delta x_{n+1}) \quad , \quad \text{dove } \Delta x_n = (x_n - x_{n-1})$$

Anche qui, la stima deve essere presa con le cautele già viste per il metodo di Newton

Metodo Lin-Bairstow

Si tratta di un metodo molto conosciuto e usato in moltissimi programmi rootfinder, apprezzato per le sue caratteristiche di praticità, efficienza e semplicità implementativa. Il metodo non necessita di aritmetica complessa pur convergendo a radici sia complesse che reali. Inoltre il metodo genera automaticamente come sottoprocesso il polinomio ridotto e tutte queste doti permettono di realizzare routine di ricerca degli zeri per polinomi reali con un codice molto compatto e lineare. Basta questo a renderlo uno dei metodi preferiti dai programmatori.

Il metodo si basa sul fatto che un polinomio $P(x)$ reale può essere rappresentato come il prodotto di polinomi lineari e quadratici. Il procedimento inizia con una stima approssimata del polinomio quadratico $D(x) = (x^2 - ux - v)$, dove "u" e "v" sono le incognite, e viene calcolato il resto della divisione fra i due polinomi $P(x) / D(x)$. In genere il resto $R(u, v)$, che è funzione delle due variabili, non sarà nullo. Il nucleo dell'algorithmo di Bairstow consiste nel procedimento iterativo di correzione dei valori "u" e "v" per avere un resto nullo. In tal modo il fattore quadratico approssimato viene trasformato in esatto, le cui radici si ottengono attraverso la formula specifica delle equazioni di 2° grado. Il processo viene quindi riapplicato al polinomio ridotto. Ad ogni iterazione viene trovate una coppia di radici e il grado del polinomio si riduce di 2. Il processo termina quando rimane un polinomio di 1° o 2° grado

In accordo a quanto detto, il polinomio originale può essere fattorizzato nella forma seguente

$$P(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n = Q(x) \cdot (x^2 - ux - v) + R(u,v)$$

dove: $R(u,v) = b_{n-1}x + b_n$, $Q(x) = b_0 x^{n-2} + \dots + b_{n-4}x^2 + b_{n-3}x + b_{n-2}$

Affinchè il fattore quadratico $(x^2 - ux - v)$ sia esatto, entrambi i coefficienti del resto devono essere nulli. Perciò si deve modificare "u" e "v" in modo che $b_{n-1} = b_n = 0$

Posto $u = u_0$, $v = v_0$, come valori iniziali

L'algorithmo inizia con le seguenti formule ricorsive:

$b_0 = a_0$	$c_0 = b_0$
$b_1 = a_1 + u b_0$	$c_1 = b_1 + u c_0$
$b_2 = a_2 + u b_1 + v b_0$	$c_2 = b_2 + u c_1 + v c_0$
.....
$b_{n-1} = a_{n-1} + u b_{n-2} + v b_{n-3}$	$c_{n-2} = b_{n-2} + u c_{n-3} + v c_{n-4}$
$b_n = a_n + u b_{n-1} + v b_{n-2}$	$c_{n-1} = b_{n-1} + u c_{n-2} + v c_{n-3}$

Gli incrementi Δu , Δv vengono calcolati mediante i valori $b_n, b_{n-1}, c_{n-1}, c_{n-2}, c_{n-3}$

$$\Delta u = \frac{b_n c_{n-3} - b_{n-1} c_{n-2}}{(c_{n-2})^2 - c_{n-1} c_{n-3}} \qquad \Delta v = \frac{b_{n-1} c_{n-1} - b_n c_{n-2}}{(c_{n-2})^2 - c_{n-1} c_{n-3}}$$

I nuovi valori sono quindi: $u + \Delta u$, $v + \Delta v$

Il processo viene ripetuto finché $|\Delta u| + |\Delta v| < \epsilon$, dove ϵ è un numero piccolo prefissato a piacere

Si osserva che i valori del vettore $\mathbf{b} = [b_0 \dots b_i \dots b_n]$ sono i coefficienti del polinomio ridotto; quindi il metodo contiene indirettamente la deflazione. quindi per ripetere il processo basta semplicemente assegnare il vettore \mathbf{b} al vettore \mathbf{a} .

Questo algoritmo si basa sull'espansione di Taylor del 2° grado applicata al polinomio $R(u, v)$; pertanto la velocità di convergenza è simile a quelle del metodo di Newton.

Prove sperimentali hanno mostrato che la convergenza non è in generale né monotonica, né regolare. Al contrario, gli incrementi Δu e Δv possono avere delle ampie oscillazioni prima di convergere.

Il costo computazionale risulta uno dei più bassi di tutti i metodi; infatti è:

(calcolo coeff. b) + (calcolo coeff. c) + (calcolo incrementi) = $4(n+2) + 4(n-6) + 13 = 8n+9$ per iterazione. Inoltre tenendo conto che ad ogni iterazione si trovano 2 radici e che il costo della deflazione è già compreso nell'algoritmo si intuisce che il metodo di Bairstow risulti essere il metodo più efficiente per la ricerca globale delle radici dei polinomi reali.

Applichiamo l'algoritmo di Bairstow all'equazione di test: $x^3 + 4x^2 + 4x + 3$, prendendo come valori iniziali $(u_0, v_0) = (0, 0)$

I coefficienti "b" e "c" sono riportati a righe alterne nella tabella seguente

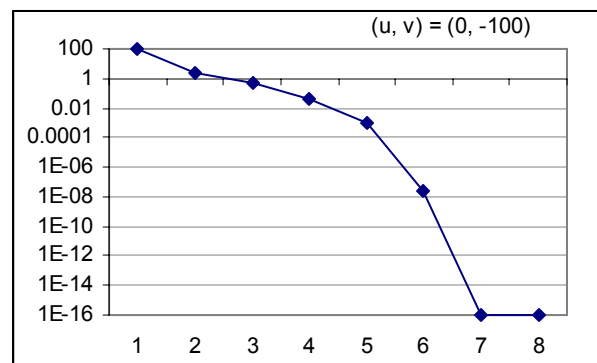
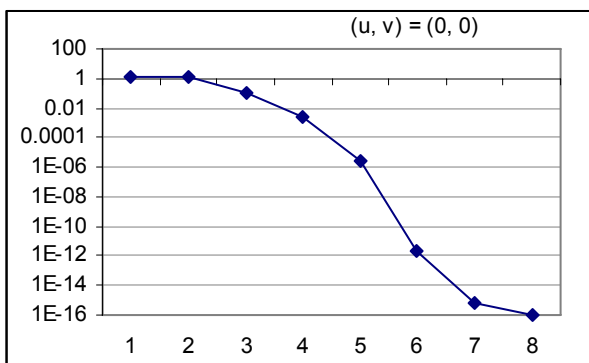
Iter		x^3	x^2	x^1	x^0	u	v	Δu	Δv	errore
	a	1	4	4	3	0	0			
1	b	1	4	4	3					
	c	1	4	4	3	-1.0833333	0.33333333	-1.08333	0.3333333	1.13E+00
2	b	1	2.91666	1.17361	2.7008					
	c	1	1.83333	-0.4791	3.83101	-0.9403255	-1.1024588	0.143008	-1.43579	1.44E+00
3	b	1	3.05967	0.0204	-0.3923					
	c	1	2.11934	-3.0748	0.16250	-0.9979133	-1.0008615	-0.05759	0.101597	1.17E-01
4	b	1	3.00208	0.00331	-0.0079					
	c	1	2.00417	-2.99753	0.97739	-0.9999989	-0.9999979	-0.00209	0.000864	2.26E-03
5	b	1	3.00000	4.35E-06	7.66E-07					
	c	1	2.00000	-2.9999	0.99999	-1	-1	-1.1E-06	-2.1E-06	2.37E-06
6	b	1	3	1.2E-12	6.0E-12					
	c	1	2	-3	1	-1	-1	4.01E-13	-2.3E-12	2.32E-12
7	b	1	3	-1.1E-15	0					
	c	1	2	-3	1	-1	-1	3.17E-16	4.76E-16	5.72E-16

In soli 7 iterazioni l'algoritmo ha trovato il fattore quadratico $(x^2 + x + 1)$ con un'ottima precisione di circa $1E-16$ e in aggiunta ha fornito anche il polinomio ridotto $(x + 3)$ nell'ultima riga di b.

Risolviendo l'equazione $x^2 + x + 1 = 0$ e $x + 3 = 0$ si trovano le radici $x_1 = -0.5 + \sqrt{3}/2$, $x_2 = -0.5 - \sqrt{3}/2$, $x_3 = -3$.

Praticamente tutte le radici sono state trovate in sole 7 iterazioni!. Il costo dell'intero processo è di circa $7(24+9) = 161$ op. La risoluzione della stessa equazione per mezzo dell'algoritmo di Newton, che pure è uno dei più efficienti, avrebbe richiesto, ipotizzando 6 iterazioni, un costo di circa: (costo ricerca radice complessa) + (costo deflazione) = $6(4(4n-1))+3n+2 = 275$ op

I grafici mostrano la traiettoria dell'errore per i valori d'innescio rispettivamente di $(0,0)$ e $(0,-100)$



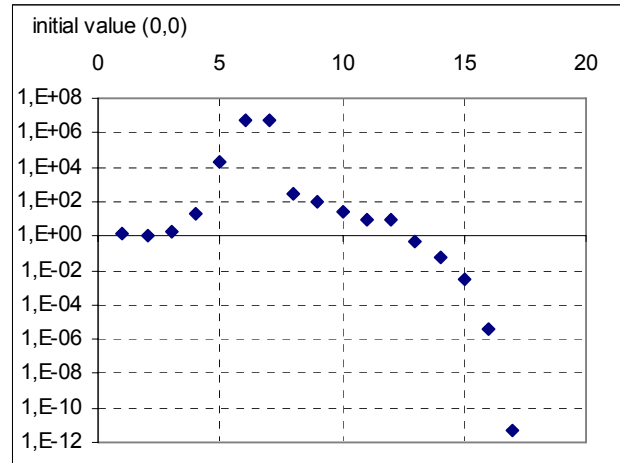
Come si vede non esiste molta differenza fra le due traiettorie: la convergenza è stata raggiunta in meno di 8 iterazioni nonostante i valori d'innescio del secondo caso siano molto lontani dalla soluzione esatta (-1, -1). Ma la convergenza è sempre così rapida ed indipendente dal punto d'innescio? Sfortunatamente no. Anche qui, come nel metodo di Newton, dipende dal polinomio e dal punto d'innescio. Anche qui la raccomandazione del "sufficientemente vicino" è molto comune in letteratura: ma per questo algoritmo, come vedremo, oltre che ambigua può diventare addirittura esilarante.

Vediamo alcuni esempi.

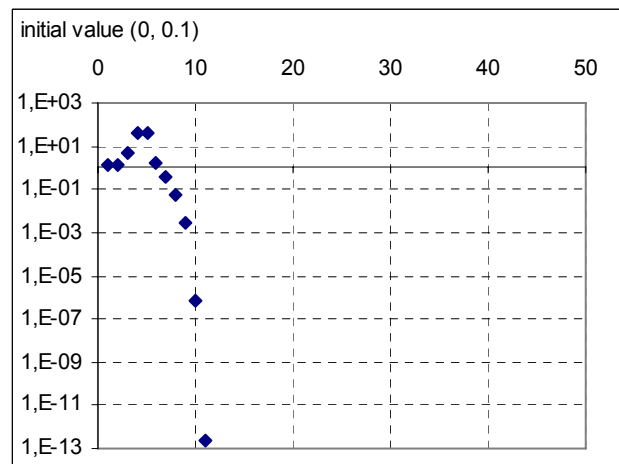
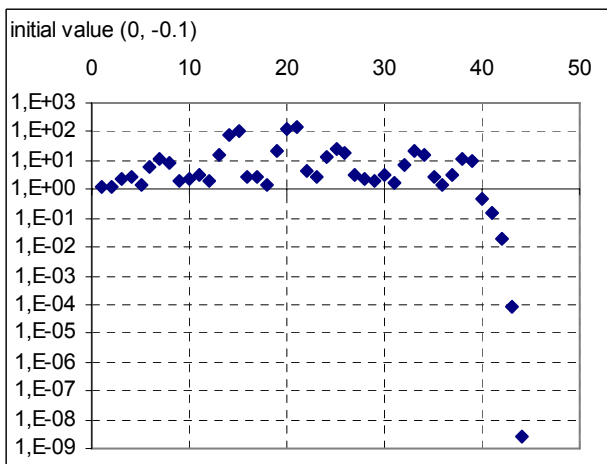
Risolviamo il polinomio $x^3 + 3x^2 + 3x + 2$ partendo, come in precedenza, dal punto d'innescio (0, 0)

In questo caso la traiettoria dell'errore è ben diversa. Dalle prime iterazione l'algoritmo comincia a divergere fin a raggiungere il massimo di circa $1E6$ alla 7 iterazione. Poi, sorprendentemente, l'algoritmo inverte bruscamente la tendenza e inizia a diminuire l'errore per convergere velocemente dopo la 15 iterazione. In pratica tutto il lavoro di approssimazione è stato fatto negli 3 passi. Il resto del tempo l'algoritmo "ha passeggiato" per il piano complesso.

Questo fenomeno è tipico di questo algoritmo la cui convergenza è persino più "selvaggia" di quello di Newton.



Questo fenomeno diventa ancor più evidente se scegliamo come punto d'innescio i valori (0, -0.1) oppure i valori (0, 0.1)

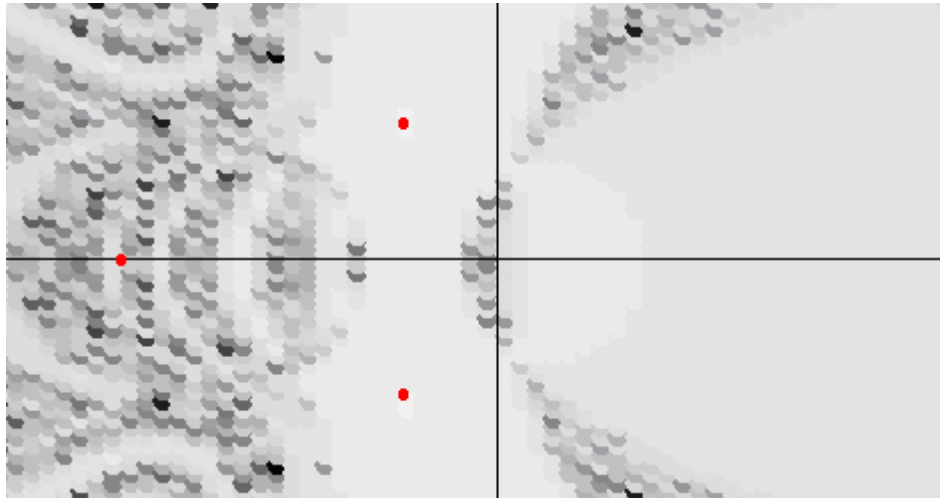


I grafici sono eloquenti. Il comportamento dell'algoritmo, pur partendo da due punti molto vicini risulta completamente diverso: nel primo caso ci vogliono più di 40 passi per trovare la convergenza, mentre nel secondo caso ne bastano solo 10. Sorprendentemente se prendessimo ancora un punto molto lontano dalla soluzione esatta come (0, -100) la convergenza si avrebbe in circa 8 passi.

Molti metodi sono stati sperimentati per correggere o perlomeno regolare la traiettoria dell'algoritmo di Bairstow attraverso una scelta oculata del punto d'innescio, ma senza molto

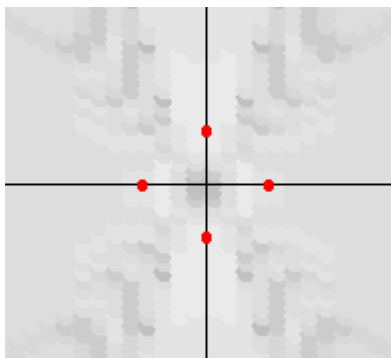
successo. Forse l'unico vero suo difetto è proprio in questa imprevedibilità del percorso di convergenza.

E' interessante vedere lo spettro di convergenza del polinomio $x^3 + 3x^2 + 3x + 2$

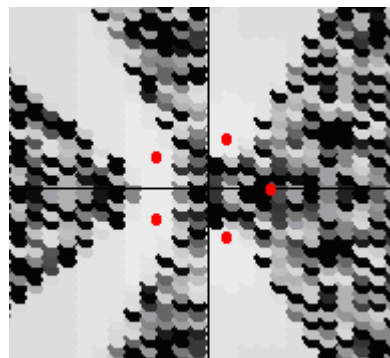


Le radici (in rosso) si trovano nei punti $(-0.5, 0.866)$ $(-0.5, -0.866)$ $(-3, 0)$. Vediamo che la coppia d'innesci $(0, -0.1)$ corrisponde ai punti $(0, 0.316)$, $(0, -0.316)$ che cadono nella zona scura vicino all'asse y, mentre la coppia d'innesci $(0, 0.1)$ corrisponde ai punti $(0.316, 0)$, $(-0.316, 0)$ che cadono invece nella zona chiara.

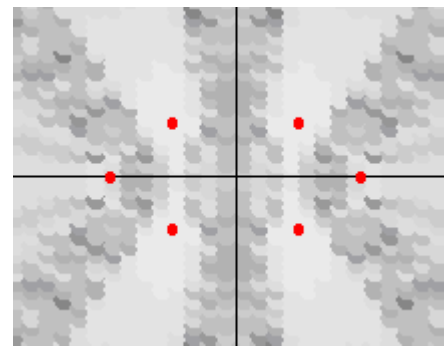
Riportiamo gli spettri di convergenza di tre differenti polinomi ottenuti con il metodo di Bairstow



$z^4 - 1$



$z^5 - 1$



$z^6 - 4z^4 + 4z^2 - 16$

A conferma di quanto avevamo anticipato, il forte contrasto fra le zone scure e chiare mette in evidenza la forte instabilità dell'algorithmo di Bairstow rispetto alle condizioni d'innesci. Osserviamo che nello spettro del polinomio $z^5 - 1$, l'unica radice reale $x = 1$ è completamente circondata da zone scure. Infatti questa radice non può essere mai estratta dall'algorithmo. Questo accade in generale quando le radici reali sono in numero dispari, poiché sappiamo che l'algorithmo di Bairstow ricerca sempre le radici a coppie. Anche in questo caso, come nel metodo di Newton, l'interno del cerchio delle radici presenta zone ad alta "turbolenza".

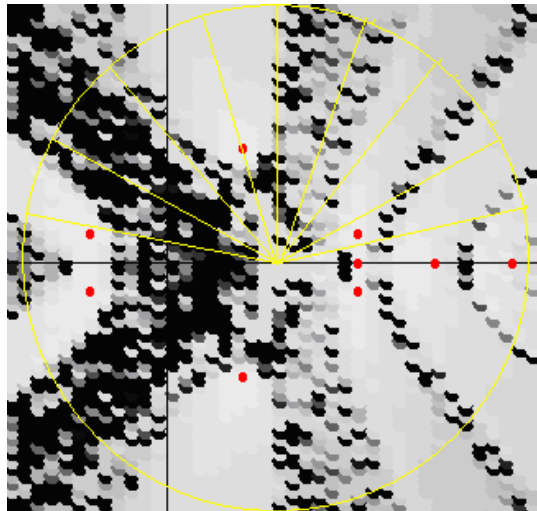
Un modo semplice e poco costoso per innescare l'algorithmo è quello di scegliere in modo casuale la coppia (u, v) nell'intervallo $-1 \leq u \leq 1$, $-1 \leq v \leq 1$. Insieme alla centratura del polinomio questo accorgimento, a meno di casi patologici, dà una ragionevole garanzia di successo, (insuccessi

minore di 5 %) per $n < 8$. Per polinomi di grado $n > 8$, la probabilità di fallimento cresce rapidamente.

Ad esempio, dato il polinomio

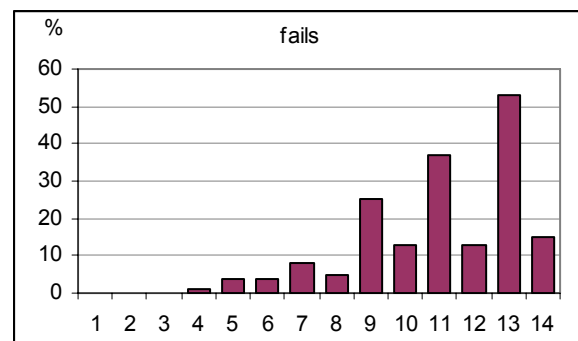
$$-819000+195400x+109270x^2-37748x^3-9169x^4+8519x^5-2510x^6+388x^7-31x^8+x^9$$

avente le radici: $-2 \pm i$, $2 \pm 4i$, $5 \pm i$, $5, 7, 9$, il suo spettro di convergenza con l'algoritmo di Bairstow è riportato, insieme al cerchio delle radici, nella figura seguente



Si osserva chiaramente la rilevante quantità di zone scure anche al di fuori del cerchio di convergenza. Questo significa una probabilità non trascurabile di fallimento della convergenza (convergence fail). Per il polinomio dato si misura una probabilità di fallimento di circa 25%.

Per gradi maggiori la condizione di convergenza è ancora più difficile. Nel diagramma seguente viene riportata la probabilità di fallimento in funzione del grado. Il rilievo statistico è stato effettuato con polinomi random di grado variabile fra $2 < n < 15$, con radici miste, complesse e reali, con $|z| < 200$. Curiosamente osserviamo il fenomeno per cui l'algoritmo converge più facilmente con i polinomi di grado pari.



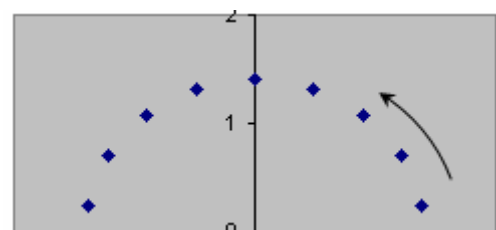
Un modo per aumentare sensibilmente la probabilità di successo senza ridurre l'efficienza del metodo è il seguente: se l'errore non diminuisce dopo un certo numero d'iterazioni (esempio 30) si effettua una sorta di attacco sistematico a "ventaglio".

Posto R = raggio delle radici, x_c = centro delle radici,
 n = grado, $\varphi = \pi/n$, si provano successivamente i punti d'innesco, per $k = 0, 1 \dots n-1$

$$x_k = R \cdot \cos(k \varphi + \varphi / 2) + x_c$$

$$y_k = R \cdot \sin(k \varphi + \varphi / 2)$$

$$u = 2 \cdot x_k \quad , \quad v = -(x_k^2 + y_k^2)$$



Partendo da ogni punto, si misura la convergenza dopo 30 iterazioni; in caso negativo si passa al punto successivo. In generale, in meno di n prove, l'algoritmo riesce a trovare un "canale" per raggiungere una radice con alta probabilità di successo (tipicamente superiore al 99%).

Metodo Siljak

Si tratta di un variante del metodo di Newton-Raphson originalmente sviluppato per i polinomi a coefficienti complessi per elaboratori sprovvisti di librerie aritmetica complessa. Grazie ad alcuni accorgimenti è risultato molto efficiente, stabile e compatto. Ne illustriamo gli aspetti principali. Si consideri il generale polinomio di grado n , scritto nella forma seguente:

$$P(x + iy) = \sum_{k=0}^n (a_k + ib_k)(x + iy)^k$$

Il polinomio può essere decomposto in due funzioni reali: $P(x + iy) = u(x, y) + i v(x, y)$

Il metodo di Siljak calcola i polinomi $u(x, y)$ e $v(x, y)$ e le derivate parziali du/dx e dv/dy attraverso i seguenti algoritmi iterativi.

Posto: $xs(0) = 1$, $ys(0) = 0$, $xs(1) = x$, $ys(1) = y$, $m = x^2 + y^2$, $t = 2x$
per $k = 2, 3, \dots, n$

$$xs(k) = t \cdot xs(k-1) - m \cdot xs(k-2)$$

$$ys(k) = t \cdot ys(k-1) - m \cdot ys(k-2)$$

I numeri $xs(k)$ e $ys(k)$, chiamati coefficienti di Siljak, sono rispettivamente la parte reale e immaginaria della potenza $(x + iy)^k$

Posto: $u = 0$, $v = 0$, per $k = 0, 1, 2, \dots, n$

$$u = u + a(k) \cdot xs(k) - b(k) \cdot ys(k)$$

$$v = v + a(k) \cdot ys(k) + b(k) \cdot xs(k)$$

Posto: $p = 0$, $q = 0$, per $k = 1, 2, \dots, n$

$$p = p + k \cdot [a(k) \cdot xs(k-1) - b(k) \cdot ys(k-1)]$$

$$q = q + k \cdot [a(k) \cdot ys(k-1) + b(k) \cdot xs(k-1)]$$

Infine, l'incremento $\Delta z = -P(z)/P'(z) = \Delta x + i \Delta y$, viene calcolato con le formule

$$\Delta x = - (u \cdot p + v \cdot q) / (p^2 + q^2) \quad , \quad \Delta y = (u \cdot q - v \cdot p) / (p^2 + q^2)$$

Le caratteristiche di velocità e stabilità sono simili al metodo di Newton. Tuttavia per aumentare la stabilità globale di convergenza, nell'algoritmo di Siljak è stato adottato l'accorgimento di controllare ad ogni iterazione la convergenza a zero del modulo $e^{(i)} = |P(z_i)|$. Se il valore $e^{(i)}$ al passo i -esimo è molto superiore di quello del passo precedente allora si dimezza l'incremento Δz e si ripete il calcolo finché il valore $e^{(i)}$ è inferiore o al limite dello stesso ordine del precedente. Questo semplice accorgimento assicura una buona stabilità. Pur non avendo prove sulla convergenza globale, il metodo ha sempre superato i test di convergenza anche usando come punto d'innescio il semplice valore fisso $(x, y) = (0.1, 1)$

Il metodo usa la deflazione per ridurre il grado del polinomio dopo ogni radice trovata

Metodo Laguerre

Un ottimo algoritmo per la ricerca degli zeri reali o complessi dei polinomi è il metodo di Laguerre. Per la sua complessità viene poco usato nel calcolo manuale ma le sue doti di robustezza, velocità e stabilità ne fanno uno degli algoritmi preferiti per il calcolo automatico.

Il metodo di Laguerre si basa sulla formula iterativa

$$z_{i+1} = z_i - \frac{n P(z_i)}{P'(z_i) \pm \sqrt{H(z_i)}} \tag{1}$$

dove $H(z)$ è :

$$H(z) = (n-1)[(n-1)P'(z)^2 - n P(z) P''(z)]$$

Il denominatore di (1) deve essere scelto in modo da minimizzare l'incremento $|dz|$; poiché in generale i due denominatori sono complessi, si sceglie quello con il modulo maggiore. La radice può essere complessa nel caso $H < 0$. In generale la formula di Laguerre richiede l'aritmetica complessa.

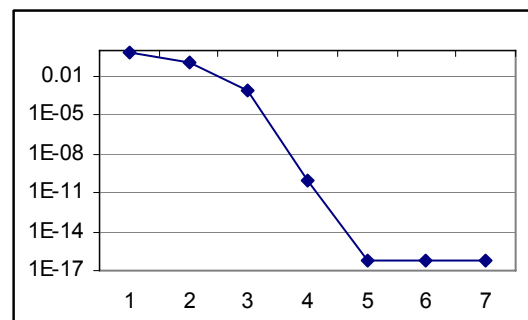
Ad esempio troviamo la radice del polinomio z^4+1 partendo da $z_0 = 1$. Si noti che il polinomio ha solo radici complesse date da $z = \pm\sqrt{2}/2 \pm i\sqrt{2}/2$, e che stiamo partendo da un numero reale puro. Nonostante ciò l'algoritmo converge alla radice complessa in meno di 5 iterazioni con un errore inferiore a $1E-16$.

Variabile.	iter 1	iter 2	iter 3	iter 4	iter 5					
x	1	0	0.8	0.6	0.707614	0.706599	0.707107	0.707107	0.707107	0.707107
P	2	0	0.1568	0.5376	4.12E-06	0.002871	0	3.7E-10	0	2.78E-16
P'	4	0	-1.408	3.744	-2.82233	2.834512	-2.82843	2.828427	-2.82843	2.828427
P''	12	0	3.36	11.52	0.017229	11.99999	2.22E-09	12	1.78E-15	12
H	-144	0	-40.32	-138.24	-0.20675	-144	-2.7E-08	-144	-1E-12	-144
√H	1.94E-14	12	7.2	-9.6	8.479188	-8.49137	8.485281	-8.48528	8.485281	-8.48528
d1	4	12	5.792	-5.856	5.656859	-5.65686	5.656854	-5.65685	5.656854	-5.65685
d2	4	-12	-8.608	13.344	-11.3015	11.32588	-11.3137	11.31371	-11.3137	11.31371
d1	12.64911		8.236504		8.000006		8		8	
d2	12.64911		15.87955		16		16		16	
d	4	12	-8.608	13.344	-11.3015	11.32588	-11.3137	11.31371	-11.3137	11.31371
dx	-0.2	0.6	-0.09239	0.106599	-0.00051	0.000508	-6.5E-11	6.54E-11	-4.9E-17	4.91E-17

La complessità del calcolo è evidente ed è stato frazionato in diverse variabili intermedie per una migliore comprensione: $d_1 = P' + \sqrt{H}$, $d_2 = P' - \sqrt{H}$, $d = \max(d_1, d_2)$, $dx = -n \cdot P/d$

Quando la radice è singola la convergenza è cubica mentre per radici multiple la convergenza è lineare.

Uno dei maggiori pregi di questo algoritmo è l'elevata stabilità della convergenza rispetto al punto d'innescio. Praticamente l'algoritmo di Laguerre converge da quasi tutti i punti del piano complesso.



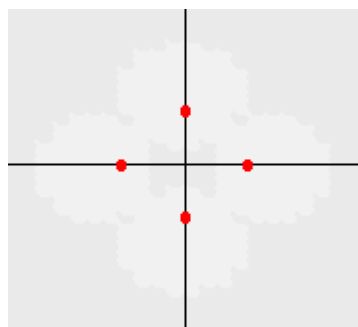
Ovviamente tutto questo va a scapito del costo computazionale che per questo algoritmo è alto. L'estrazione della radice quadrata complessa è un'operazione che richiede le funzioni trascendenti goniometriche: il costo della radice da sola può essere stimato a circa 5 volte il costo di una

moltiplicazione complessa, cioè 20 op. Inoltre si conteggiano il calcolo del polinomio complesso e delle sue derivate prima e seconda, oltre alle normali operazioni aritmetiche complesse. In totale il costo per iterazione di un polinomio di grado "n" è di circa $12(2n+3)$.

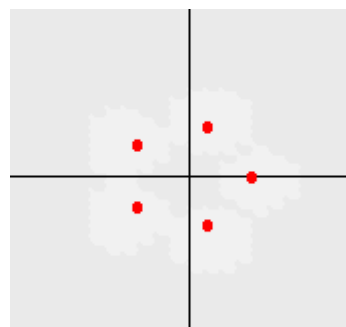
Nel caso precedente, $n = 4$, il costo totale per 5 iterazioni è stato di $12(2n+3)5 = 660$; con il metodo di Newton si ottiene lo stesso risultato in 9 iterazioni al costo complessivo di $(16n-4)9 = 540$, mentre con il metodo di Bairstow si ha 10 iterazioni al costo di $(8n+9)10 = 410$. Probabilmente il costo è l'unico punto sfavorevole di questo metodo.

La convergenza globale invece, come abbiamo accennato, è ottima.

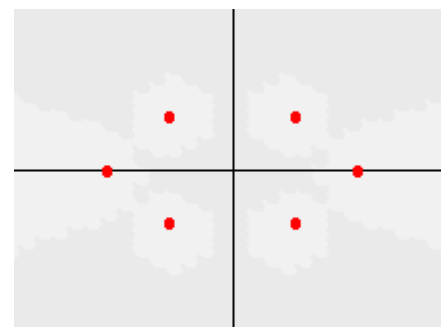
Riportiamo gli spettri di convergenza di tre differenti polinomi ottenuti con il metodo di Laguerre



z^4-1



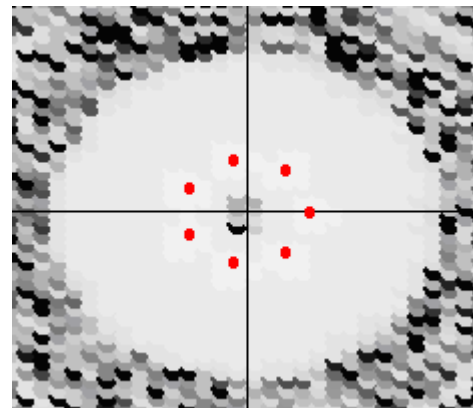
z^5-1



$z^6-4z^4+4z^2-16$

Come si nota ogni punto d'innescio scelto a caso sul piano complesso converge alla radice in pochissime iterazioni (< 8)

Tuttavia da prove sperimentali è emerso che la scelta dei punti d'innescio nella corona circolare compresa fra 1 e 2 volte il raggio delle radici è ancora la migliore. Infatti se tracciamo lo spettro di convergenza del polinomio x^7-1 , osserviamo zone di "turbolenza" per il cerchio $r > 3$, e, se pur in misura ridotta, anche all'interno del cerchio delle radici, in prossimità dell'origine



Metodo a convergenza simultanea (ADK)

Questo metodo ha la caratteristica di convergere simultaneamente a tutte le radici del polinomio. Si basa sulla formula iterativa di Newton-Rapson applicata alla funzione razionale

$$F_i(z) = \frac{P(z)}{\prod_{j=1}^{i-1} (z - z_j^{(0)}) \cdot \prod_{j=i+1}^n (z - z_j^{(0)})}$$

dove $z_j^{(0)}$ sono delle approssimazioni delle radici del polinomio $P(z)$. La funzione $F_i(z)$ ha gli stessi zeri del polinomio, ma in pratica converge solo alla i -esima radice z_i , in quanto i poli, molto vicini alle radici diverse da z_i , impediscono l'avvicinamento di qualunque successione.

Lo schema iterativo, veramente geniale, è stato sviluppato e migliorato da numerosi autori ed è noto sotto vari nomi quali metodo di Aberth-Durand-Kerner o formula iterativa di Herlich o di Mahely o anche metodo di Weierstrass.

La variante più veloce prevede il calcolo del polinomio e della sua derivata. Se indichiamo con $z_i^{(k)}$ la radice i -esima al passo k , e $\Delta_i^{(k)} = P(z_i^{(k)}) / P'(z_i^{(k)})$ la formula iterativa è:

$$z_i^{(k+1)} = z_i^{(k)} + \frac{\Delta_i^{(k)}}{1 + c_i^{(k)} \cdot \Delta_i^{(k)}} \quad , \quad i = 1, 2 \dots n \quad (1)$$

Dove il fattore correttivo $c_i^{(k)}$ è dato da:

$$c_i^{(k)} = \sum_{j=1}^{i-1} \frac{1}{z_i^{(k)} - z_j^{(k+1)}} + \sum_{j=i+1}^n \frac{1}{z_i^{(k)} - z_j^{(k)}}$$

Si osserva che se il termine correttivo $c_i \cong 0$ la formula ritorna quella di Newton

La formula iterativa (1) viene innescata con un vettore $\mathbf{z}^{(0)} = [z_1^{(0)}, z_2^{(0)}, z_3^{(0)} \dots z_n^{(0)}]$ formato con le migliori approssimazioni disponibili delle n radici del polinomio.

Dalla prima iterazione si ricava il valore $z_1^{(1)}$, che sostituisce $z_1^{(0)}$ nel vettore; successivamente si ricava $z_2^{(1)}$ che sostituisce $z_2^{(0)}$ e così via. In pratica il ciclo d'iterazioni segue lo schema seguente

$$\begin{aligned} [z_1^{(0)}, z_2^{(0)}, z_3^{(0)} \dots z_{n-1}^{(0)}, z_n^{(0)}] &\Rightarrow z_1^{(1)} \\ [z_1^{(1)}, z_2^{(0)}, z_3^{(0)} \dots z_{n-1}^{(0)}, z_n^{(0)}] &\Rightarrow z_2^{(1)} \\ [z_1^{(1)}, z_2^{(1)}, z_3^{(0)} \dots z_{n-1}^{(0)}, z_n^{(0)}] &\Rightarrow z_3^{(1)} \\ [z_1^{(1)}, z_2^{(1)}, z_3^{(1)} \dots z_{n-1}^{(0)}, z_n^{(0)}] &\Rightarrow z_4^{(1)} \\ &\dots\dots\dots \\ [z_1^{(1)}, z_2^{(1)}, z_3^{(1)} \dots z_{n-1}^{(1)}, z_n^{(0)}] &\Rightarrow z_n^{(1)} \end{aligned}$$

A questo punto si sono ottenute tutte le radici relative al primo passo, cioè si è ottenuto il nuovo vettore $\mathbf{z}^{(1)}$ e si può ricominciare il ciclo

$$\begin{aligned} [z_1^{(1)}, z_2^{(1)}, z_3^{(1)} \dots z_{n-1}^{(1)}, z_n^{(1)}] &\Rightarrow z_1^{(2)} \\ &\dots\dots\dots \end{aligned}$$

Dopo un certo numero di cicli d'iterazioni la sequenza vettoriale $\{\mathbf{z}^{(k)}\}$ converge simultaneamente a tutte le radici del polinomio reali e/o complesse. Nel caso si sappia che tutte le radici sono reali non è nemmeno necessario usare l'aritmetica complessa

Sebbene non ci siano prove matematiche, la stabilità globale di questo metodo è molto alta. Prove sperimentali hanno mostrato che il metodo può convergere anche partendo da approssimazioni iniziali molto lontane.

Scelta del vettore iniziale.

Un metodo molto semplice consiste nel prendere una distribuzione circolare di n punti centrata nel centro delle radici e avente raggio $\rho < r < 2\rho$

Praticamente per un polinomio generico di grado "n", si ricava il centro $C = (x_c, 0)$ e il raggio delle radici ρ con uno dei metodo precedentemente visti, quindi si pone

$$\theta = 2\pi / n,$$

$$q = 1/2 \text{ se } n \text{ pari, } q = 1 \text{ se } n \text{ dispari}$$

$$\text{per } i = 1, 2, \dots, n, \quad z_i = (\rho \cos(\theta \cdot (i - q)) + x_c, \rho \sin(\theta \cdot (i - q)))$$

Ad esempio dato il polinomio

$$z^6 - 18z^5 + 131z^4 - 492z^3 + 1003z^2 - 1050z + 425$$

Il centro del polinomio è dato da $x_c = 18/6 = 3$. Facendo la traslazione $z = w+3$, otteniamo il polinomio centrato: $w^6 - 4w^4 + 4w^2 - 16$,

che ha le seguenti radici $w = -1 \pm i, w = 1 \pm i, w = -2, z = 2$

Quindi le radici del polinomio originale sono $z = 2 \pm i, w = 4 \pm i, w = 1, z = 5$

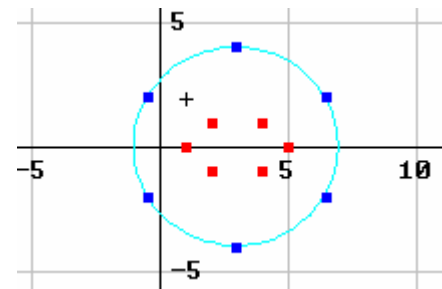
Vediamo come si sceglie il vettore iniziale per l'innescò dell'algoritmo ADK

Calcoliamo la stima del raggio delle radici

$$\rho = 2 \max [4^{(1/2)}, 4^{(1/4)}, 16^{(1/6)}] = 4$$

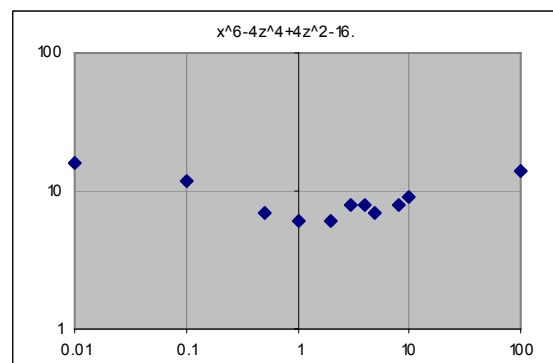
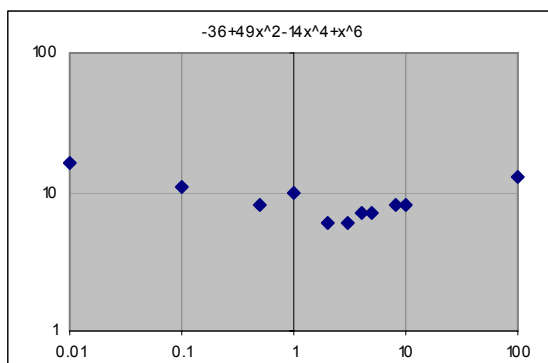
$$z_i = (4\cos(\pi/3 \cdot (i - 1/2)) + 3, 4\sin(\pi/3 \cdot (i - 1/2))), \quad i = 1, 2, \dots, 6$$

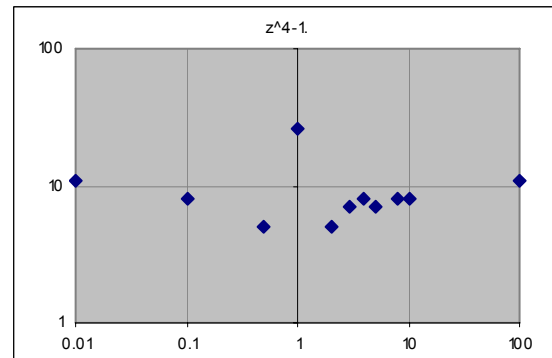
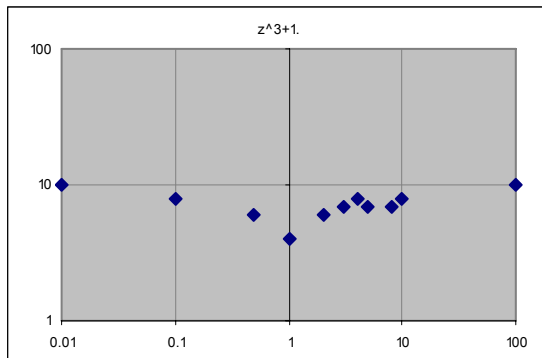
Le radici (rosso) e i punti iniziali (blu) so riportati nel grafico a sinistra



Partendo da questi valori il metodo ADK converge contemporaneamente alle 6 radici in circa 8 cicli, con una precisione di circa 1E-16. Ricordiamo che ogni ciclo corrisponde a 6 iterazioni.

Viene da interrogarsi su quale sarebbe stato il comportamento se avessimo preso un cerchio iniziale differente. Nei seguenti grafici abbiamo riportato l'andamento del numero di cicli in funzione del raggio del cerchio iniziale scelto nell'intervallo $0.01 < r < 100$, per differenti polinomi di test





Come si vede il comportamento è estremamente stabile per quasi tutti i valori del raggio del cerchio iniziale. Chiaramente la scelta dei valori d'innescio non è un problema di questo algoritmo.

Vediamo ora il costo computazionale.

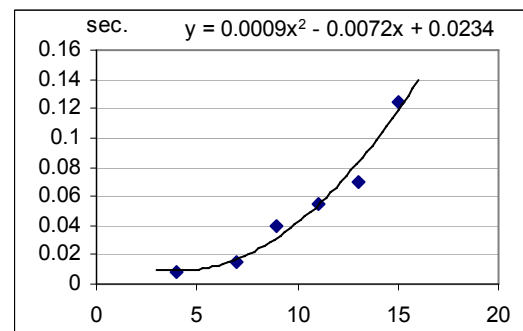
Calcolo formula (1) in aritmetica complessa:	16n+8
Calcolo fattore correttivo in aritmetica complessa:	3n-3
Totale per iterazione:	19n+5
Totale per ciclo:	$(19n+5)n = 19n^2+5n$

Ad esempio per il polinomio di 6° grado dell'esempio precedente il costo totale sarebbe stato di circa 5700, contro i 3000 del metodo di Newton-Rapshon.

Come si vede il costo rimane molto alto, poiché questo metodo non trae beneficio dalla riduzione del grado apportata dalla deflazione.

Nel grafico sono riportati i tempi (s) rilevati per la risoluzione di alcuni polinomi comuni di grado compreso fra 4 e 15.

Come si vede i tempi rimangono sotto il decimo di secondo per i polinomi fino al grado < 15. Il test è stato effettuato su un comune PC, Pentium, 1.8GHz, con programma in VBA.



Nonostante la sua bassa efficienza, questo metodo ha trovato larga applicazione nelle routine automatiche¹ per l'eliminazione definitiva degli errori della deflazione e per le caratteristiche di accuratezza, stabilità, robustezza e, ultimo ma non meno importante, per la facilità implementativa

¹ Ottime implementazioni basate sul metodo di Aberth si trovano in: MPSolve, un potente programma rootfinder, sviluppato in ANSI C da D. A. Bini e G. Fiorentino all'Univeristà di Pisa "PRECISE: Efficient Multiprecision Evaluation of Algebraic Roots", Shankar Krishnan, et al. (AT&T Labs - Research)

Zeri dei polinomi ortogonali

Il metodo NR

Un metodo molto usato per l'approssimazione di tutti gli zeri dei polinomi ortogonali è composto dai seguenti passi

Approssimazione iniziale di una radice
 x_i^*

Questo compito è affidato a diverse formule specializzate: in genere sono usate le approssimazioni polinomiali, razionali, o anche espansioni in serie di potenze

Calcolo del polinomio e sua derivata
 $L(x), L'(x)$

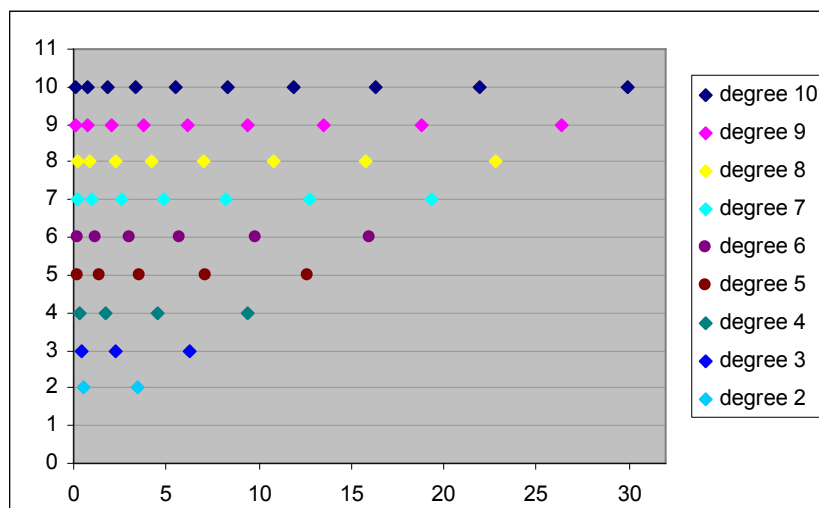
Questo compito è eseguito per mezzo di formule ricorsive capaci di calcolare simultaneamente il polinomio e la sua derivata prima. Queste formule non hanno bisogno dei coefficienti del polinomio e sono molto efficienti ed accurate

Root refinement with Newton-Raphson method
 $x_{n+1} = x_n - L(x_n)/L'(x_n)$

Il raffinamento della radice è effettuato per mezzo del metodo iterativo di Newton-Raphson. Se l'approssimazione iniziale è sufficientemente vicina la convergenza è molto rapida ed accurata.

La chiave del successo di questo schema è nel primo passo. Se la stima iniziale x_i^* della radice non è sufficientemente accurata il metodo può convergere ad una altra radice non voluta, o già trovata. Questo può dare dei problemi specialmente quando il grado del polinomio diventa elevato perchè alcune radici diventano sempre più vicine

La figura seguente mostra gli zeri dei polinomi di Laguerre dal 2° al 10° grado.



Come si nota, la distanza fra le radici incrementa a destra ma diminuisce a sinistra. La distanza fra x_1 e x_2 diviene molto piccola per i polinomi di alto grado. Per approssimare queste radici, l'algoritmo iterativo NR necessita di un valore d'innescò molto preciso altrimenti non riesce a trovare tutte le radici del polinomio. In genere le radici più piccole sono le più difficili da approssimare.

Questo fenomeno spiega perchè in generale le formule di stima sono così complicate.

Ad esempio delle buone formule¹ di stima per i polinomi generalizzati di Laguerre $L(n, m)$ sono le seguenti

$$z_1 = \frac{(1+m)(3+0.92m)}{1+2.4n+1.8m} \quad z_2 = z_1 + \frac{15+6.25m}{1+0.9n+2.5m}$$

$$z_i = z_{i-1} + \frac{\left(\frac{1+2.55(i-2)}{1.9(i-2)} + \frac{1.26(i-2)m}{1+3.5(i-2)} (z_{i-1} - z_{i-2}) \right)}{1+0.3m}$$

La cui implementazione in VB è

```

If i = 1 Then
    z(1) = (1 + m) * (3 + 0.92 * m) / (1 + 2.4 * n + 1.8 * m)
ElseIf i = 2 Then
    z(2) = z(1) + (15 + 6.25 * m) / (1 + 0.9 * m + 2.5 * n)
Else
    z(i) = z(i - 1) + ((1 + 2.55 * (i - 2)) / (1.9 * (i - 2)) + 1.26 * (i - 2) *
        m / (1 + 3.5 * (i - 2))) * (z(i - 1) - z(i - 2)) / (1 + 0.3 * m)

```

Decisamente piuttosto complicate! A compenso di ciò, queste formule sono molto veloci ed accurate e permettono al metodo NR di approssimare tutte le radici dei polinomi di Laguerre. Fortunatamente nella letteratura specializzata sono note anche le formule per altri famiglie di polinomi ortogonali (Legendre, Hermite, Chebychev, Jacobi).

Il metodo ADK

Un metodo alternativo per l'approssimazione di tutti gli zeri dei polinomi ortogonali e quello cosiddetto della "deflazione implicita" che usa cioè la formula ADK²

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n) - p(z_n) \sum_{j=1}^r \left(\frac{1}{z_n - z_j} \right)}$$

dove $I_r = \{z_1, z_2, \dots, z_r\}$ è un sottoinsieme delle radici. La formula iterativa convergerà ad una radice non contenuta nell'insieme I_r . Le cose funzionano come se la radici fossero state "estratte" dal polinomio stesso attraverso la deflazione. Questo impedisce di fatto la convergenza verso queste radici e spiega il termine "deflazione implicita". Questa formula non richiede, come vedremo, una precisa stima della radice iniziale, ma solo una grossolana limitazione superiore. Di conseguenza non necessita di mettere punto nessuna complicata formula di approssimazione iniziale.

In aggiunta osserviamo che il metodo è valido per ogni famiglia di polinomi ortogonali. o, meglio per tutti i polinomio aventi tutte le radici reali.

Vediamo ora come funziona con un esempio pratico

Strategia di attacco per i polinomi ortogonali

Assumiamo di ricercare le radici del polinomio di Laguerre del 4° grado

$$L_4(z) = \frac{1}{24} z^4 - \frac{2}{3} z^3 + 3z - 4z + 1$$

¹ ("Numerical recipes in Fortran77", Cambridge U Press, 1999)

² ADK sta per Aberth-Durand-Kerner, ma la formula è attribuita anche a Mahely e ad Ehrlich

Sappiamo che tutte le radici sono singole e reali e positive

$$0 < z_1 < z_2 < z_3 < z_4 < R$$

Il limite superiore può essere grossolanamente approssimato per $n \leq 20$ con la semplice funzione

$$R \cong 3.5 \cdot (n - 1)$$

o con una più accurata

$$R \cong 3.14 \cdot (n - 1) + 0.018 \cdot (n - 1)^2$$

All'inizio l'insieme delle radici I_r è vuoto ($r = 0$) e la formula ADK ritorna quella di Newton

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n)}$$

Se scegliamo come punto d'innescio x_0 un numero uguale o maggiore (ma non troppo) di $R = 10.5$, la formula iterativa convergerà alla massima radice x_4

$$x_4 \cong 9.39507091230113$$

Osserviamo che il punto d'innescio non è affatto critico. L' algoritmo converge ugualmente bene anche per $x_0 = 12, 15$, ecc. Sottolineiamo solo che non deve essere preso troppo distante dalla massima radice perchè, in generale, i polinomi ortogonali crescono molto rapidamente al di fuori dell'intervallo delle radici creando problemi di overflow.

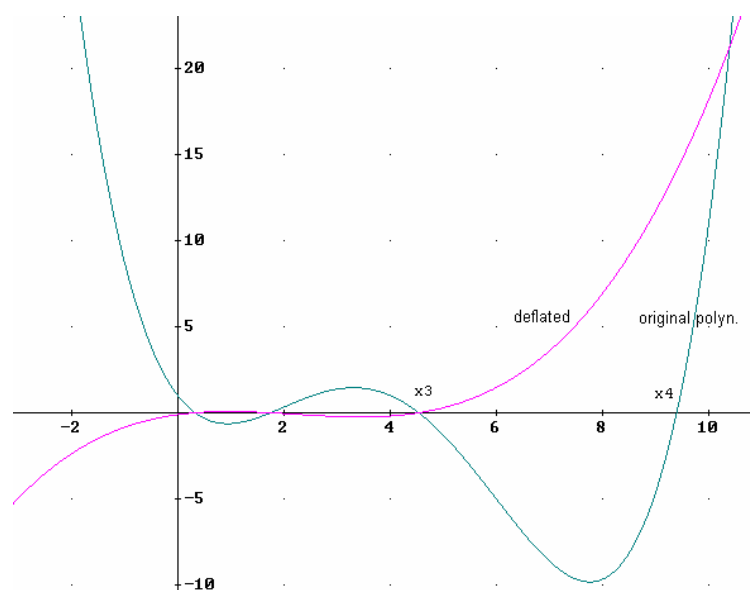
Quando la massima radice è stata trovata la formula ADK cambia in

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n) - p(z_n) \frac{1}{z_n - z_4}}$$

Ora il nuovo punto d'innescio può essere scelto vicino a x_4 e maggiore di x_3 , che è la massima radice del polinomio ridotto $L_{d4}(x)$

$$L_{d4}(z) = \frac{L_4(z)}{z_n - z_4}$$

La situazione è mostrata nel grafico seguente.



Osserviamo che non dobbiamo calcolare effettivamente il polinomio ridotto $L_{d4}(x)$ per mezzo della deflazione. La formula ADK semplicemente "salta" la radice x_4

Un nuovo punto d'innescò che soddisfa la condizione $x_0 > x_3$ è sicuramente $x_0 = x_4 + h$, dove "h" è un valore piccolo a piacere (esempio 5% of x_4). Questo accorgimento è necessario perchè la formula non può partire da una radice esatta a causa della divisione per zero (1/0 error)

Sostituendo, otteniamo il nuovo valore d'innescò $x_0 \cong 9.4 + 0.47 = 9.87$; da cui la convergenza alla radice

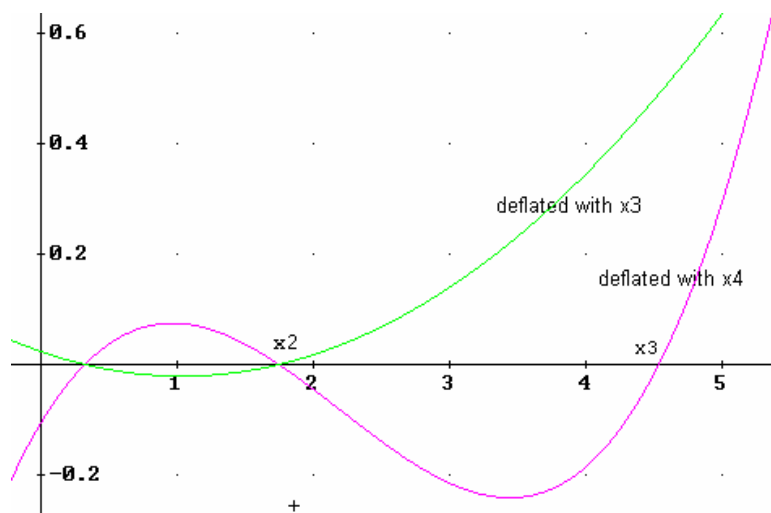
$$x_3 \cong 4.53662029692113$$

Di nuovo inserendo la radice x_3 nella formula ADK otteniamo

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n) - p(z_n) \left(\frac{1}{z_n - z_4} + \frac{1}{z_n - z_3} \right)}$$

e un altro punto d'innescò $x_0 \cong 4.5 + 0.225 = 4.725$, che converge alla radice

$$x_2 \cong 1.74576110115835$$



Inserendo anche la radice x_2 nella formula ADK, otteniamo

$$z_{n+1} = z_n - \frac{p(z_n)}{p'(z_n) - p(z_n) \left(\frac{1}{z_n - z_4} + \frac{1}{z_n - z_3} + \frac{1}{z_n - z_2} \right)}$$

Con il punto iniziale $x_0 \cong 1.75 + 0.05 = 1.785$, la formula converge all'ultima radice

$$x_1 \cong 0.322547689619392$$

Riassumendo tutte le radici del polinomio di Laguerre di 4° grado sono

$x_4 =$	9.39507091230113
$x_3 =$	4.53662029692113
$x_2 =$	1.74576110115835
$x_1 =$	0.32254768961939

Ora è chiaro come funziona il metodo ADK e il motivo per cui l'accuratezza del punto d'innescio iniziale non è così importante come nel metodo di Newton.

Limitazione della massima radice
 $x_n < R$

Questo compito è svolto per mezzo di semplici formule. L'accuratezza non è richiesta

Calcolo del polinomio e della derivata
 $L(x), L'(x)$

Questo compito è eseguito per mezzo di formule ricorsive capaci di calcolare simultaneamente il polinomio e la sua derivata prima. Queste formule non hanno bisogno dei coefficienti del polinomio e sono molto efficienti ed accurate

Approssimazione delle radici con il metodo ADK

Questo compito è svolto dalla formula iterativa ADK. Se il punto d'innescio è maggiore della massima radice la convergenza è garantita per tutte le radici.

Osserviamo inoltre che le radici sono sempre trovate in ordine decrescente; questo è utile per risparmiare un ulteriore passo di riordinamento delle radici

Metodo Jenkins-Traub

Si tratta di un di un metodo ibrido, cioè composto da vari algoritmi che permettono il raggiungimento della convergenza globale per ogni tipo di polinomio, conservando nel contempo una buona efficienza media. La prima versione di questo eccellente rootfinder, diventato ormai uno standard, è stata sviluppata in FORTRAN 4. La versione aggiornata si trova nella libreria IMSL con il nome "CPOLY" (polinomi complessi) o "RPOLY" (polinomi reali) o in Mathematica¹ con il nome "Nsolve"

Il funzionamento di questo programma è molto complesso. La versione in FORTRAN richiede quasi 1000 linee di codice e circa 700 linee la versione tradotta in C++.

Moltissimi accorgimenti sono stati inglobati per assicurare la corretta convergenza ed evitare errori di overflow e di arrotondamento.

Quello che vogliamo illustrare in queste pagine sono solo i concetti generali e il cuore dell'algoritmo. Le radici sono calcolate una per volta, partendo da quelle reali di più basso modulo. Dopo che una radice è stata trovata il polinomio viene ridotto per deflazione e l'algoritmo è riapplicato al polinomio ridotto.

L'algoritmo JT si basa sulla ricerca di opportuni polinomi $H(z)$ di grado $n-1$ aventi le stesse radici z_2, z_3, \dots, z_n , del polinomio originale $P(z)$, meno quella di modulo minore z_1 . Il quoziente $P(z)/H(z)$ sarà quindi una funzione lineare avente come unica radice z_1

Ovviamente il polinomio $H(z)$ non viene trovato esattamente ma attraverso una procedura iterativa e di conseguenza, fermando l'iterazione ad una certo passo k , le radici di $H_k(z)$ non saranno esattamente la radici di $P(z)$ ma (si spera) molto vicine a queste. I poli della funzione razionale $P(z)/H_k(z)$ tenderanno a scoraggiare la convergenza verso tutte le radici eccetto z_1 , che può essere trovata mediante una processo iterativo tipo punto unito.

Il processo iterativo concettuale per trovare i polinomi $H_k(z)$, a meno di speciali accorgimenti, è il seguente. Posto

$$H(z) = h_{n-1} \cdot z^{n-1} + h_{n-2} z^{n-2} \dots + h_2 \cdot z^2 + h_1 \cdot z + h_0 +$$

Posto $H_0(z) = P'(z)$

$$H_{k+1}(z) = \frac{1}{z} \left(H_k(z) - \frac{H_k(0)}{P(0)} P(z) \right) , k = 1, 2 \dots K$$

In generale bastano pochi passi ($K = 4 \div 8$) per avere un soddisfacente polinomio

Vediamo un esempio. Prendiamo il polinomio $P(z) = z^3 - 8z^2 + 17z - 10$ avente tutti le radici reali

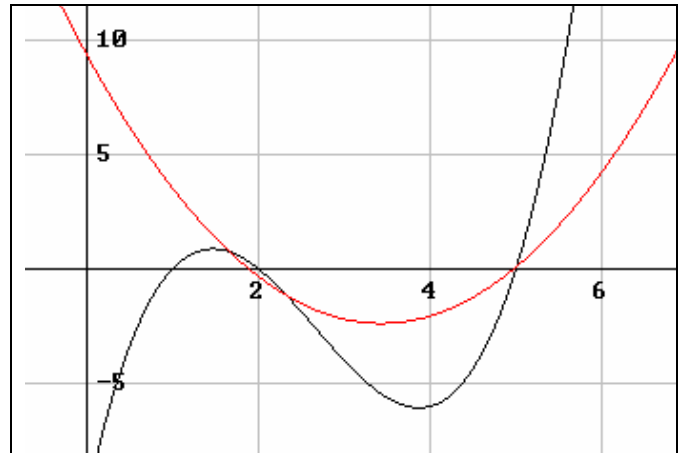
$$\begin{aligned} H_0(z) &= 17 - 16z + 3z^2 \\ H_1(z) &= 12.9 - 10.6z + 1.7z^2 \\ H_2(z) &= 11.33 - 8.62z + 1.29z^2 \\ H_3(z) &= 10.641 - 7.774z + 1.133z^2 \end{aligned}$$

In questo caso specifico la successione di polinomi $\{H_k\}$ è formata da parabole che convergono alla parabola avente i due zeri uguali alle radici del polinomio dato di più alto modulo

¹ Mathematica , WolframResearch

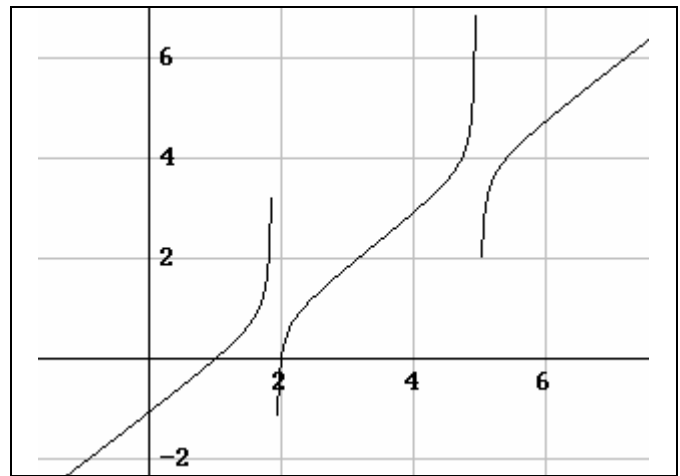
Nel grafico a fianco sono stati tracciati i grafici del polinomio e della parabola $H_3(z)$. Si vede chiaramente che la parabola incontra l'asse x nei punti molto vicini agli zeri del polinomio dato.

Procedendo nella successione i punti diventano sempre più vicini agli zeri del polinomio. Tuttavia gli zeri di H_3 , e quindi i poli di $1/H_3$, sono in genere sufficienti a favorire la convergenza verso la radice più piccola. (JT normalmente calcola fino a H_5)



Il processo di ricerca della radice viene effettuato con il metodo del punto unito applicato alla funzione razionale $P(z) / H_3(z)$, riportata ne grafico,

Si vede chiaramente che la funzione ha un andamento lineare eccetto in prossimità dei due punti $x = 2$ e $x = 5$, probabile locazione della radici più grandi. Osserviamo che nell'intorno della radice più piccola $x = 1$, l'andamento della funzione è lineare.



Questa trasformazione è il cuore dell'algorithmo di JT in cui si tende ad evidenziare la radice più piccola neutralizzando simultaneamente tutte le radici più grandi per mezzo di opportuni poli localizzati nelle immediate vicinanze.

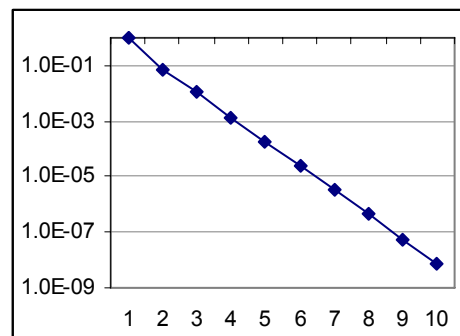
Lo schema iterativo JT per la ricerca della radice minore è quindi:

$$z_{i+1} = z_i - \frac{P(z_i)}{\tilde{H}_k(z_i)}$$

dove \tilde{H} è il polinomio monico di H : $\tilde{H}_k(z_i) = \frac{H_k(z)}{h_{n-1}}$

Innescando la formula iterativa con $x_0 = 0$ otteniamo il seguente schema

x	P	\tilde{H}	dx	dx
0	-10	9.3918799	1.0647496	1.0647496
1.0647496	0.2383072	3.2198672	-0.0740115	0.0740115
0.9907381	-0.0374774	3.5755621	0.0104816	0.0104816
1.0012196	0.004871	3.5245225	-0.001382	0.001382
0.9998376	-0.0006498	3.5312397	0.000184	0.000184
1.0000216	8.639E-05	3.5303451	-2.447E-05	2.447E-05
0.9999971	-1.149E-05	3.530464	3.255E-06	3.255E-06
1.0000004	1.528E-06	3.5304482	-4.329E-07	4.329E-07
0.9999999	-2.033E-07	3.5304503	5.757E-08	5.757E-08
1	2.703E-08	3.5304501	-7.657E-09	7.657E-09



La convergenza, con andamento lineare alla radice $z_1 = 1$ è evidente.

L'ordine di convergenza in questo caso è 1.

Il metodo JT risolve in modo brillante questo problema con il metodo della traslazione (shifting). E' stato dimostrato che, con questo accorgimento, l'ordine di convergenza è equivalente a quello di Newton- Raphson. Vediamo come funziona.

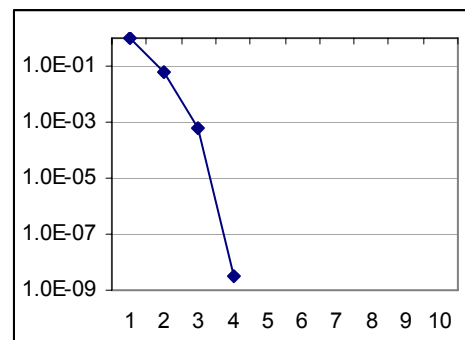
Nella seconda fase del processo, la formula iterativa dei polinomio H_k si modifica in:

$$z_{i+1} = z_i - \frac{P(z_i)}{\tilde{H}_k(z_i)}$$

$$H_{k+1}(z) = \frac{1}{z - z_{i+1}} \left(H_k(z) - \frac{H_k(z_{i+1})}{P(z_{i+1})} P(z) \right)$$

In questo modo la traslazione è inserita nel processo iterativo in modo stabile ed efficiente.

x	P	\tilde{H}	dx	dx
0	-10	9.39188	1.06474955	1.0647496
1.0647496	0.2383072	3.715299	-0.0641421	0.0641421
1.0006074	0.0024279	3.996942	-0.0006074	0.0006074
1	-1.28E-08	3.999979	3.2092E-09	3.209E-09
1	6.75E-14	3.999979	-1.688E-14	1.688E-14



La convergenza è aumentata in modo considerevole. In soli 4 iterazioni abbiamo raggiunto una precisione di circa 1E-14

Analogamente a quanto accade nel metodo di Newton, per approssimare ad una radice complessa il processo iterativo deve essere innescato con un valore complesso; diversamente, il metodo non può convergere ad una radice complessa

Nel programma RPOLY ci sono subroutine specializzate per controllare la presenza di radici complesse e per scegliere di conseguenza il punto d'innescio più appropriato. Altre funzioni sono adibite al controllo generale della convergenza del processo, ed altre ancora a limitare la propagazione degli errori di arrotondamento. Una particolare attenzione è stata posta al controllo della presenza o meno delle radici "cluster" che, come abbiamo mostrato rallentano la convergenza analogamente alle radici multiple. Elencare dettagliatamente tutte questi accorgimenti implementativi va al di là del nostro scopo.

Costo computazionale. Nel corso degli anni il metodo JT è stato costantemente migliorato in termini di efficienza e, nonostante l'elevato numero di linee di codice, il tempo macchina impiegato nella risoluzione completa di un polinomio risulta in assoluto un dei più bassi. Risulta di gran lunga meno costoso dell' algoritmo di Laguerre o di Durand-Kerner e, se pur in misura più ridotta, perfino di quello di Newton; solo il metodo di Bairstow, quando converge, può competere in termini di efficienza con quello di Jenkins-Traub.

Metodo QR

Un altro metodo per la ricerca simultanea di tutte le radici di un polinomio si basa sugli autovalori della matrice compagna.

Dato un polinomio monico (cioè con $a_n = 1$) si compone la seguente matrice

$$a(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1} + x^n$$

$$A = \begin{bmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -a_{n-1} \end{bmatrix}$$

Gli autovalori di $[A]$ sono, come noto, le radici del polinomio. Quindi possiamo adoperare tutti i metodi matriciali di ricerca degli autovalori per trovare le radici di un qualunque polinomio.

Uno dei metodi più usati sfrutta l'algoritmo di fattorizzazione QR, dove Q è una matrice ortogonale e R una matrice triangolare superiore. Concettualmente l'algoritmo è semplice e si basa sul seguente schema iterativo.

Posto $A_0 = A$, si decompone la matrice A_0 nel prodotto $Q_1 \cdot R_1$ e si costruisce la matrice $A_1 = R_1 \cdot Q_1$; si ripete la decomposizione di A_1 e così via. La successione di matrici $\{A_n\}$ converge ad una matrice triangolare superiore i cui elementi della diagonale principale possono essere valori reali, nel caso di autovalori reali, o sottomatrici 2×2 nel caso di autovalori complessi. Possono verificarsi, ad esempio, i seguenti due casi

λ_1	#	#	#	#	#
0	λ_2	#	#	#	#
0	0	λ_3	#	#	#
0	0	0	λ_4	#	#
0	0	0	0	λ_5	#
0	0	0	0	0	λ_6

λ_1	#	#	#	#	#
0	λ_2	#	#	#	#
0	0	α_{11}	α_{12}	#	#
0	0	α_{21}	α_{22}	#	#
0	0	0	0	β_{11}	β_{12}
0	0	0	0	β_{21}	β_{22}

Gli autovalori della matrice di destra sono tutti reali. La matrice di sinistra ha 2 autovalori reali (λ_1 e λ_2) e 2 coppie di autovalori complessi coniugati nelle sottomatrici 2×2 " α " e " β ".

Per ottenerli basta risolvere i polinomi caratteristici delle due sottomatrici.

$$\lambda^2 - (\alpha_{11} + \alpha_{22}) \lambda + (\alpha_{11} \cdot \alpha_{22} - \alpha_{21} \cdot \alpha_{12}) = 0$$

$$\lambda^2 - (\beta_{11} + \beta_{22}) \lambda + (\beta_{11} \cdot \beta_{22} - \beta_{21} \cdot \beta_{12}) = 0$$

Il metodo QR per la ricerca degli autovalori richiede il calcolo matriciale.

Per contro ha due importantissimi vantaggi: non richiede alcun valore d'innesco ed è intrinsecamente stabile e robusto. Sebbene non vi siano teoremi che affermano la sua globale convergenza, la pratica ha mostrato che questo algoritmo converge quasi sempre.

Queste caratteristiche l'hanno reso molto popolare ai programmatori già da molti anni: HQR della libreria Fortran Eispak, è infatti una delle migliori routine di ricerca degli autovalori per matrici reali standard

Esempio. Radici reali

Dato il polinomio $x^3 - 8x^2 + 17x - 10$ costruiamo la matrice compagna e ricerchiamo gli autovalori con la fattorizzazione QR¹.

$$\begin{array}{|c|c|c|} \hline \text{A0} & & \\ \hline 0 & 0 & 10 \\ \hline 1 & 0 & -17 \\ \hline 0 & 1 & 8 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline \text{Q1} & & \text{R1} \\ \hline 0 & 0 & 1 \\ \hline 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 1 & 0 & -17 \\ \hline 0 & 1 & 8 \\ \hline 0 & 0 & 10 \\ \hline \end{array}$$

Moltiplicando R₁ per Q₁ si ottiene la nuova matrice A₁ a cui si riapplica la fattorizzazione

$$\begin{array}{|c|c|c|} \hline \text{A1} & & \\ \hline 0 & -17 & 1 \\ \hline 1 & 8 & 0 \\ \hline 0 & 10 & 0 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline \text{Q2} & & \text{R2} \\ \hline 0 & -0.862 & 0.507 \\ \hline 1 & 0 & 0 \\ \hline 0 & 0.507 & 0.8619 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 1 & 8 & 0 \\ \hline 0 & 19.723 & -0.862 \\ \hline 0 & 0 & 0.507 \\ \hline \end{array}$$

Dopo 6 iterazioni vediamo già la convergenza di A₆ ad una matrice triangolare e la comparsa degli autovalori sulla diagonale principale

$$\begin{array}{|c|c|c|} \hline \text{A6} & & \\ \hline 5.1028 & -17.04 & 11.435 \\ \hline 0.0186 & 1.9466 & -1.773 \\ \hline 0 & 0.0277 & 0.9506 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline \text{Q7} & & \text{R7} \\ \hline 1 & -0.004 & 5E-05 \\ \hline 0.0036 & 0.9999 & -0.014 \\ \hline 0 & 0.0138 & 0.9999 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 5.1029 & -17.03 & 11.428 \\ \hline 0 & 2.0088 & -1.801 \\ \hline 0 & 0 & 0.9755 \\ \hline \end{array}$$

Proseguendo, dopo 20 iterazioni si ottiene la matrice A₂₀

$$\begin{array}{|c|c|c|} \hline \text{A20} & & \text{iter. } 20 \\ \hline 5 & -16.74 & 11.881 \\ \hline 5E-08 & 2 & -1.871 \\ \hline -3E-16 & 2E-06 & 1 \\ \hline \end{array}$$

Osserviamo due fenomeni: l'annullamento progressivo dei valori al di sotto della diagonale principale e la convergenza di questa ai valori $\lambda_1 = 5, \lambda_2 = 2, \lambda_3 = 1$. Da questo si deduce che tutti e tre gli autovalori sono reali e di conseguenza anche le radici del polinomio dato. Il massimo valore residuo al di sotto della diagonale, (2E-6) da anche una stima dell'errore degli autovalori.

Esempio. Radici complesse coniugate

Dato il polinomio $x^4 - 7x^3 + 21x^2 - 37x + 30$ costruiamo la matrice compagna e ricerchiamo gli autovalori. Con 100 iterazioni del metodo QR otteniamo la matrice A₁₀₀ seguente

$$\begin{array}{|c|c|c|c|} \hline \text{A0} & & & \\ \hline 0 & 0 & 0 & -30 \\ \hline 1 & 0 & 0 & 37 \\ \hline 0 & 1 & 0 & -21 \\ \hline 0 & 0 & 1 & 7 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|} \hline \text{A100} & & & \\ \hline 3 & -2.91838 & -25.1713 & -44.6849 \\ \hline 9.02E-14 & -0.33519 & -4.67545 & -7.93471 \\ \hline 2.72E-21 & 1.236835 & 2.335192 & 4.270452 \\ \hline 3.34E-16 & -6.8E-22 & 6.8E-06 & 2 \\ \hline \end{array}$$

Osserviamo due fenomeni: l'annullamento progressivo dei valori al di sotto della diagonale principale solo per i valori $\lambda_1 = 3, \lambda_4 = 2$. e la convergenza dei valori della matrice in generale. Da questo si deduce che gli autovalori λ_2 e λ_3 sono complessi coniugati

¹ La fattorizzazione QR può essere ottenuta facilmente in Excel per mezzo dell'aggiunta Matrix.xla (by Foxes Team)

Per calcolarli esplicitamente dobbiamo ricavare il polinomio caratteristico $a_2\lambda^2 + a_1\lambda + a_0$ della submatrice matrice (2x2)

-0.33519	-4.67545
1.236835	2.33519

a2	a1	a0
1	-2	5.0000

Risolvendo l'equazione associata, si ha: $\lambda^2 - 2\lambda + 5 = 0 \Rightarrow \lambda = 1 \pm 2i$

Esempio. Radici multiple

Dato il polinomio $x^4 - 7x^3 + 17x^2 - 17x + 6$ costruiamo la matrice compagna e ricerchiamo gli autovalori. Con 100 iterazioni del metodo QR otteniamo la matrice A_{100} seguente

A0			
0	0	0	-6
1	0	0	17
0	1	0	-17
0	0	1	7

 \Rightarrow

A100			
3	-13.8636	16.82906	-12.6343
4.77E-15	2	-3.41298	1.872992
2.28E-16	3.85E-15	1.010308	-0.89431
-1E-16	3.83E-16	0.000119	0.989692

Osserviamo due fenomeni: l'annullamento progressivo dei valori al di sotto della diagonale principale solo per i valori $\lambda_1 = 3, \lambda_2 = 2$. e la convergenza dei valori della matrice in generale. Da questo si deduce che gli autovalori λ_3 e λ_4 potrebbero essere complessi coniugati. Per calcolarli esplicitamente dobbiamo ricavare il polinomio caratteristico $a_2\lambda^2 + a_1\lambda + a_0$ della submatrice matrice (2x2)

1.010308	-0.89431
0.000119	0.989692

a2	a1	a0
1	-2	1

Risolvendo l'equazione associata, si ha: $\lambda^2 - 2\lambda + 1 = 0 \Rightarrow (\lambda - 1)^2 = 0 \Rightarrow \lambda = 1, m = 2$

Quindi, in realtà, non abbiamo due valori complessi ma una radice doppia. Le radici multiple, anche in questo algoritmo, rallentano la convergenza. Tuttavia usando questo accorgimento si può sovente ottenere dei valori accurati senza incrementare troppo il numero delle iterazioni.

Per accelerare la convergenza e rendere più efficiente il metodo QR sono stati escogiatati numerosi e ingegnosi accorgimenti quali, ad esempio, la tecnica dello "shifting". E' stato mostrato, infatti, che la convergenza a zero degli elementi al di sotto della diagonale segue la relazione: $a_{ij}^{(n)} \approx (\lambda_i / \lambda_j)^n$. Di conseguenza, la convergenza può essere molto lenta se λ_i e λ_j sono molto vicini. Effettuando la trasformazione dalla matrice $\mathbf{B} = \mathbf{A} - k\mathbf{I}$, dove "k" una costante reale o complessa opportuna, anche gli autovalori vengono traslati della stessa quantità. Però in questo caso la convergenza dei coefficienti di \mathbf{B} è data da: $b_{ij}^{(n)} \approx [(\lambda_i - k) / (\lambda_j - k)]^n$. L'idea per aumentare la velocità di convergenza è quella di scegliere "k" in modo da rendere massimo il rapporto: Una buona scelta è quella di scegliere, ad ogni iterazione, un valore prossimo all'autovalore più piccolo o ad una sua stima. Notare che con lo "shifting" gli autovalori possono apparire sulla diagonale principale in ordine qualsiasi, mentre senza questa tecnica gli autovalori appaiono in ordine decrescente di valore assoluto.

Test per polinomi

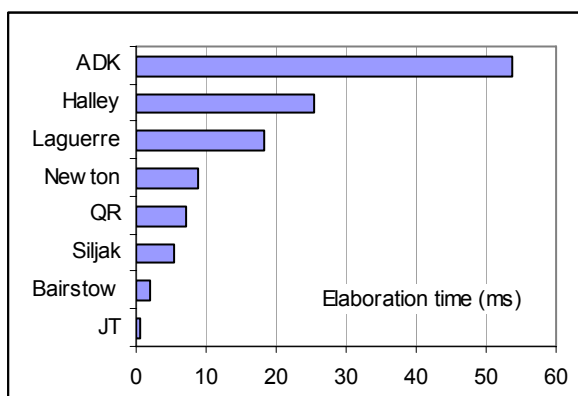
Test Random

La misura del costo computazionale e quindi dell'efficienza globale di un metodo rootfinding per polinomi non può essere affrontato in modo analogo a quello per le funzioni. Gli algoritmi per polinomio contengono molte linee di codice: dalle centinaia dei più compatti alle migliaia di quelli ibridi. Di conseguenza il conteggio esatto di ogni singola operazione sarebbe molto oneroso. Inoltre sarebbe anche inutile in quanto in generale siamo interessati alla misura del costo computazionale nella sua globalità cioè per la ricerca di tutte le radici del polinomio e non per una singola radice. Il tempo totale di calcolo dipende da molti fattori: dal grado del polinomio; dalla distribuzione spaziale delle sue radici e dalla loro molteplicità; dai metodi e dai valori d'innesco; dalla precisione richiesta e perfino dalla sequenza di estrazione delle radici. Tutti questi fattori non possono essere conosciuti a priori per cui l'approccio migliore sembra quello statistico.

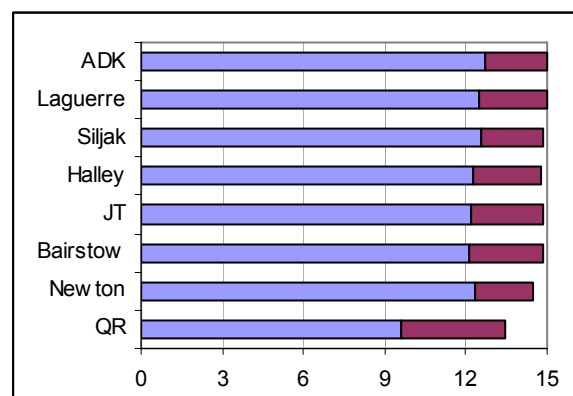
Il campione statistico. sono stati generati in modo casuale 1100 polinomi di grado progressivamente crescente da 3 a 14, aventi radici miste reali e complesse, con modulo $|z| < 100$. Per ogni gruppo di 100 polinomio di grado uguale sono stati rilevati il tempo di calcolo¹, l'errore medio delle radici (LRE)² e la deviazione standard dell'errore. E' stato rilevato anche il numero dei casi di non convergenza (fails).

Il riepilogo dei valori medi è riportato nella tabella e nei grafici seguenti

Metodo	tempo (ms)	LRE	Dev. LRE	fails	start
JT	0.68	13.5	1.32	0%	-
Bairstow	1.87	13.5	1.38	0.2%	random + fun
Siljak	5.37	13.7	1.16	0%	fisso
QR	7.23	11.5	1.95	0%	-
Newton	8.97	13.4	1.05	0%	cerchio delle radici
Laguerre	18.16	13.8	1.32	0%	cerchio delle radici
Halley	25.36	13.5	1.28	0%	cerchio delle radici
ADK	53.66	14.0	1.22	0%	cerchio delle radici



Tempo di elaborazione medio (ms)



LRE medio

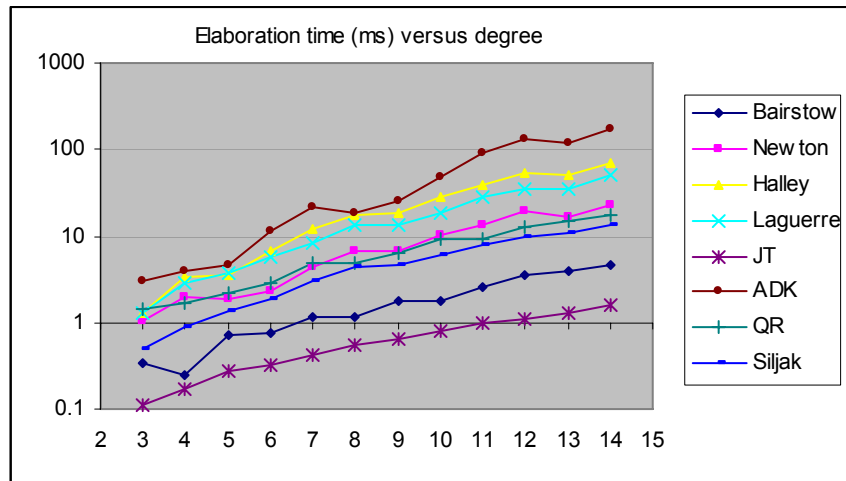
Come si vede praticamente tutti gli algoritmi hanno superati il test di convergenza (solo due errori per l'algoritmo di Bairstow). Il metodo "ibrido" di Jenkins-Traub (JT), a dispetto delle sue quasi 1000 linee di codice è risultato velocissimo e molto preciso. I metodi a convergenza cubica

¹ Il tempo è stato ottenuto con una processore Intel Pentium, 1.2GHz

² LRE : (Log. relative error) è definito $LRE = \min(-\text{Log}_{10}(e), 15)$

(Laguerre e Halley) risentono della loro maggiore complessità elaborativa. Il metodo ADK, non effettuando la deflazione, è risultato il più lento ma per contro è risultato molto accurato. L'errore LRE medio è stato di circa 13.3 (± 1.3).

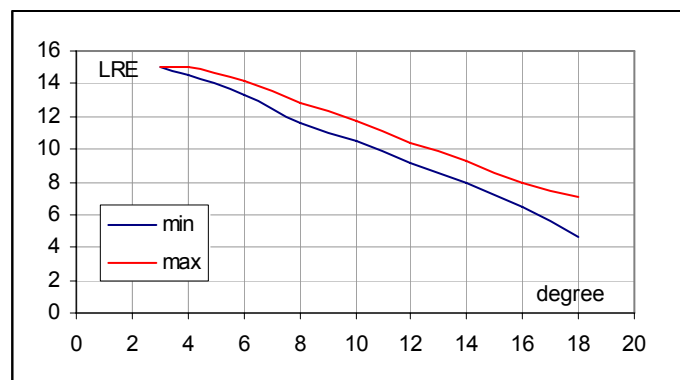
Nel grafico seguente sono riportati, per ogni algoritmo, i tempi medi di elaborazione in funzione del grado del polinomio



Test Wilkinson

La famiglia dei polinomi avente tutte le radici intere $x = \{1, 2, 3 \dots n\}$, è molto usata per provare la precisione degli algoritmi rootfinding. Questi polinomi, studiati da Wilkinson ed altri, sono molto fastidiosi per quasi tutti i metodi. Applichiamo quindi i metodi visti ai polinomi di grado $3 \leq n \leq 18$ rilevando la precisione media LRE. I calcoli sono stati effettuati in precisione standard (15 cifre)

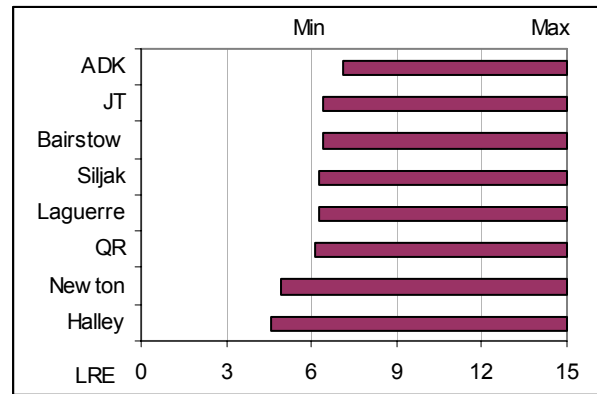
n	LRE min	LRE medio	LRE max
3	15	15	15
4	14.5	14.9	15
6	13.3	13.7	14.2
8	11.6	12.2	12.8
10	10.5	11.2	11.7
12	9.2	10	10.4
14	7.9	8.7	9.3
16	6.5	7.2	8
18	4.6	6	7.1



Come si nota, la precisione ottenuta per i polinomi di W. è molto minore di quella dei del precedente test "random". Per i polinomi "random" di grado 14 avevamo ottenuto LRE = 13.3 mentre in questo test abbiamo ottenuto LRE = 8.7, più di 4 ordini di grandezza inferiore. Però il test ci evidenzia anche un sostanziale equilibrio fra i vari algoritmi e un comportamento abbastanza omogeneo in funzione del grado. Si fa osservare che i polinomi di W. con $n > 16$ possono essere scritti solo in modo approssimativo poiché alcuni coefficienti superano le 15 cifre significative.

Il comportamento globale dei singoli algoritmi nei confronti dei polinomi di W. è sintetizzato nel seguente grafico, in cui sono riportati il max e min LRE.

Il leggero vantaggio del metodo ADK è dovuto principalmente al fatto le radici vengono trovate simultaneamente e quindi viene evitata la propagazione degli errori di arrotondamento generati dalla deflazione.



Test di selettività

In letteratura si trovano molte "test-suit" di polinomi¹ specializzati per saggiare le caratteristiche di la stabilità, efficienza e precisione degli algoritmi rootfinder.

In genere questi polinomi "standard" sono scelti per provare la capacità di ricerca di radici multiple o la capacità di riconoscimento di radici molto vicine per polinomi di grado crescente (fino a diverse centinaia). In alcuni casi i polinomi vengono approssimati anche nei coefficienti per saggiare la robustezza degli algoritmi.

Fra le centinaia di polinomi di test, il seguente polinomio, estratto dal Jenkins-Traub test, ci sembra valido per saggiare la selettività degli algoritmi nei confronti delle radici molto vicine.

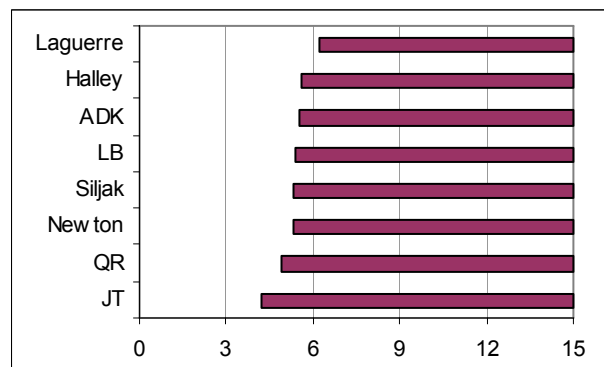
$$jt06 \quad (x-0.1)(x-1.001)(x-.998)(x-1.00002)(x-.99999)$$

Tali radici "quasi multiple" sono frequenti nei polinomi incontrati nelle scienze applicate. Il polinomio di test ha 4 radici molto vicine, fra 3E-5 a 2E-4. Se applichiamo gli algoritmi rootfinding a questo polinomio otteniamo un degrado della precisione proprio come se le radici fossero multiple.

Le grafico riportata il max e min LRE, per ogni algoritmo applicato al polinomio jt06.

Ovviamente i valori minimi si hanno per le radici vicine ad 1, mentre il massimo LRE = 15 si ottiene per la radice $x = 0.1$

Il valore minimo LRE è compreso fra 4 e 6; questo significa che la minima differenza fra due radici rilevabile dagli algoritmi in precisione standard di 15 cifre può essere compresa fra 10^{-4} e 10^{-6}



¹ Jenkins et al. 1975, Loizou 1983, Brugnano et al. 1985, Dvorkuc 1969, Farmer et al. 1975, Goedecker 1984, Igarashi et al. 1995, Iliev 2000, Miyakoda 1989, Petkovic 1989, Stolan 1995, Toh et al 1994, Uhlig 1995, Zeng 2003, Zhang 2001

Strategia di ricerca delle radici dei polinomi

Strategia d'attacco

Esistono differenti approcci per la ricerca delle radici dei polinomi, dipendendo da vari fattori come il grado del polinomio, la precisione richiesta, la scala dei coefficienti, la disposizione topologica delle radici, la loro molteplicità, ecc.. Inoltre una grande differenza è rappresentata dal dominio di ricerca: tutto il piano complesso o solo l'asse reale.

Vediamo in questo capitolo come si può procedere per ottenere le radici di un generico polinomio reale in modo da ottenere la massima precisione possibile..

Il processo si articola nei seguenti passi

1. Ricerca delle eventuali radici intere. Per tali radici, come abbiamo visto, esistono metodi per estrarle virtualmente senza errori, indipendentemente dalla loro molteplicità. La deflazione intera consente di abbassare il grado senza introdurre errori di arrotondamenti
2. Ricerca di eventuali radici multiple tramite MCD. Se diverso da 1, si procede alla decomposizione del polinomio in fattori di grado minore aventi ciascuno radici singole. Il processo di decomposizione in fattori non è in generale esente da errori, ma il guadagno globale di precisione, come abbiamo mostrato, è generalmente rilevante
3. Ricerca delle radici reali dei polinomi rimanenti per mezzo delle routine rootfinder generali viste nei capitoli precedenti. Nei casi particolari si può procedere ad una ricerca selettiva delle radici (esempio solo radici reali) attraverso la scelta di opportuni valori d'inesco

Naturalmente, oltre alla strategia delineata si può applicare ad ogni polinomio, se necessario, tutti i metodi di trasformazione già visti quali la traslazione, il cambio di scala, la riduzione dei coefficienti, l'eliminazione o l'arrotondamento dei decimali , ecc...

Esempi di risoluzione di polinomi

Esempio 1. Dato il polinomio¹ di grado 9.

$$P = [200, 140, 222, 181, 46, 43, 26, 3, 2, 1]$$

Procediamo alla ricerca di eventuali radici intere con il metodo di Ruffini²

Si trova la radice $x = -2$ con molteplicità 3; effettuando la divisione per $(x+2)$, tre volte di seguito si trova il polinomio ridotto di grado 6.

$$P_1 = [25, -20, 39, -24, 15, -4, 1]$$

Effettuando la decomposizione con il metodo MCD, si ottiene

$$P_3 = [5, -2, 6, -2, 1] \quad , \quad P_3 = [5, -2, 1]$$

Si osserva che il primo polinomio P_3 è il MCD fra il polinomio P_1 e la sua derivata

¹ Negli esempi seguenti, per chiarezza, i polinomi possono essere dati o attraverso il vettore dei loro coefficienti o per mezzo della loro forma algebrica a seconda della forma più conveniente. I coefficienti sono scritti in ordine di grado crescente da sinistra a destra o dall'alto in basso.

² Un rootfinder con questo metodo si trova in Xnumbers.xla con il codice "Rootfinder RF"

I polinomi P_2 e P_3 hanno tutte radici singole che possono essere trovate con un metodo rootfinder generale¹

Il risultato finale delle radici ottenute è

Dal polinomio P	polinomio P_2	polinomio P_3
-2, -2, -2	$1 \pm 2i$, $\pm i$	$1 \pm 2i$

Tutte le radici sono state ottenute con la massima precisione² di $1E-16$

Vediamo ora cosa otteniamo se applichiamo direttamente un algoritmo rootfinder al polinomio originale $P(x)$.

L'errore sulla radice -2 è ora di circa $1.3E-5$, mentre la radice doppia $1 \pm 2i$ ha un errore di circa $2E-6$.

La radice singola $\pm i$ ha invece la massima precisione.

La precisione globale è quindi ancora buona ma certamente non comparabile con quella ottenuta con la strategia precedente

z re	z im	err
-2.000009686	1.67763E-05	1.323E-05
-2.000009686	-1.67763E-05	1.323E-05
-1.999980628	0	9.686E-06
0	1	6.172E-19
0	-1	6.172E-19
0.999999056	1.999996904	2.02E-06
0.999999056	-1.999996904	2.02E-06
1.000000944	2.000003096	2.02E-06
1.000000944	-2.000003096	2.02E-06

Se invece consideriamo la precisione rapportata al tempo impiegato, il metodo diretto, in termini di efficienza, ritorna superiore. Ci son casi tuttavia in cui gli effetti degli errori sono così catastrofici che i valori ritornati dal metodo rootfinder diretto sono privi di significato. In tal caso La strategia di attacco descritta è l'unico mezzo adatto per ottenere dei risultati accettabili.

Un altro metodo di attacco per le radici multiple è l'uso di algoritmi rootfinder che adottano il "drill-down"³. In tal caso è possibile ottenere tutte le radici con la massima precisione di $1E-16$ in modo molto efficiente.

Esempio 2.

$$P = [3888, -40176, 174096, -413028, 587577, -514960, 272600, -80000, 10000]$$

Il polinomio non ha radici intere, come prova la ricerca con il metodo di Ruffini. Vediamo se ci sono radici multiple. La scomposizione fattoriale esiste ed è la seguente:

$$P_1 = [36, -216, 443, -360, 100] \quad , \quad P_2 = [-18, 63, -64, 20] \quad , \quad P_3 = [-6, 5]$$

L'ultimo polinomio è lineare è quindi la sua radice è immediata: $x = 6/5 = 1.2$

Le radici degli altri due polinomi possono essere approssimate con l'algoritmo rootfinder

Dal polinomio P_1	Dal polinomio P_2	Dal polinomio P_3
0.4, 0.5, 1.2, 1.5	0.5, 1.2, 1.5	1.2

Quindi le radici sono: 0.4, 0.5 ($m = 2$), 1.2 ($m = 3$), 1.5 ($m = 2$)

La precisione è dell'ordine di $1E-15$

¹ In questi esempi useremo il rootfinder JT (Jenkins-Traub), se non diversamente specificato

² In questi esempio useremo la precisione standard di 15 cifre, se non diversamente specificato

³ In Xnumbers.xla l'algoritmo nominato "Rootfinder GN" (Generalized Newton-Raphson)

Se applichiamo direttamente l'algoritmo rootfinder al polinomio originale P(x) otteniamo i seguenti valori approssimati

La radice singola e quelle doppie conservano una buona precisione ma la radice 1.2 tripla degrada fino a circa 2E-4.

z re	z im	err
0.4	0	7.22E-15
0.4999999999	0	1.38E-09
0.5000000001	0	1.38E-09
1.199941593	0.00010111	0.00016
1.199941593	-0.00010111	0.00016
1.200116814	0	0.000117
1.499997919	0	2.08E-06
1.500002081	0	2.08E-06

Esempio 3. Decomporre con il metodo MCD la forma normale del polinomio¹

$$P(x) = (x - 4)^2 \cdot (x - 3)^4 \cdot (x - 2)^6 \cdot (x - 1)^8$$

La forma normale del polinomio dato è riportata nella colonna a sinistra. Nelle colonne destra, come in uno foglio elettronico, sono riportati i coefficienti dei polinomi fattori²

P(x)	P1	P1	P2	P2	P3	P3	P4	P4
82944	24	24	-6	-6	2	2	-1	-1
-1064448	-50	-50	11	11	-3	-3	1	1
6412608	35	35	-6	-6	1	1		
-24105792	-10	-10	1	1				
63397936	1	1						
-123968128								
186963852								
-222655300	Il polinomio puo essere fattorizzato come:							
212617033	P = (P1· P2· P3· P4) ²							
-164382924								
103450582								
-53083024								
22168911	Le radici di ogni polinomio, tutte singole, sono:							
-7494136								
2030608	P1		P2		P3		P4	
-434244	1, 2, 3, 4		1, 2, 3		1, 2		1	
71575								
-8764								
750								
-40								
1								
	0.4531							

Quindi, mettendo tutto insieme, troviamo le seguenti radici: 1 (m = 8), 2 (m = 6), 3 (m = 4), 4 (m = 2). Osserviamo che tutte le radici sono intere, quindi potevano essere estratte facilmente anche con il metodo di Ruffini.

¹ Chiaramente la forma fattorizzata è stata data solo per semplicità. Nei casi reali questa forma non è mai conosciuta e dobbiamo sempre partire dalla forma polinomiale normale.

² Questo è effettivamente l'output della macro "Polynomial Factors" di Xnumbers.xla

Esempio 4.

Approssimare le radici del polinomio i cui coefficienti sono riportati nella tabelle a fianco
 Come si nota fra i coefficienti esiste una grande variazione di scala: si va dal valore 1 a 10^{-21} del termine noto. I coefficienti intermedi hanno scale via via decrescenti.

grado	coeff.
0	1E-21
1	-1.11111E-15
2	1.122322110E-10
3	-1.12333211E-06
4	0.00112232211
5	-0.111111
6	1

In tali casi può essere conveniente ai fini della precisione, equalizzare i coefficienti per mezzo di una cambiamento di scala. Posto $x = y / k$, il polinomio diventa

$$P(y) = y^6 + k \cdot a_5 y^5 + k^2 \cdot a_4 y^4 + k^3 \cdot a_3 y^3 + k^4 \cdot a_2 y^2 + k^5 \cdot a_1 y + k^6 \cdot a_0$$

Il valore di k non è affatto critico. Scegliendo $k = 1000$ i coefficienti diventano quelli riportati nella tabella seguente a sinistra. Risolvendo con il metodo di Newton otteniamo le radici y_i , che divise per 1000 danno successivamente le radici originali x_i .

n	k^n	$k^n a_n$
0	1.E+18	0.001
1	1.E+15	-1.11111
2	1.E+12	112.232211
3	1.E+09	-1123.333211
4	1.E+06	1122.32211
5	1.E+03	-111.111
6	1	1

y	x	x (err. rel.)
0.000999999999968	9.99999996774E-07	3.22574E-10
0.010000000000000	0.0000100000000000	1e-16
0.09999999999986	9.99999999986E-05	1.42464E-12
1.000000000000047	0.0010000000000005	4.6946E-13
10	0.01	1e-16
100	0.1	1e-16

Come si nota la precisione relativa è molto buona

Vediamo invece cosa succederebbe se si risolvesse direttamente il polinomio dato con un algoritmo come quello di Newton¹

Come si vede, il valori più piccoli sono affetti da un grande errore relativo, progressivamente decrescente con le radici più grandi

x	x (rel. err)
9.5559951020E-07	0.04440
1.0049360690E-05	0.00494
9.9994989752E-05	5.010E-05
0.00100000005010	5.010E-08
0.00999999999995	4.960E-12
0.1	0

¹ L'algoritmo JT effettua automaticamente l'equalizzazione dei coefficienti per cui il cambio di scala non è così necessario come in altri algoritmi

Bibliografia

"Numerical Recipes in Fortran", W.H. press et al., Cambridge University Press, 1992

"Numerical Analysis" F. Sheid, McGraw-Hill Book Company, New-York, 1968

"Numerical Methods that Work", Forman S. Acton

"Iterative Methods for the solution of Equations", J. F. Traub, Prentice Hill, 1964

"Numerical Mathematics in Scientific Computation", Germud Dahlquist, Åke Björck, 2005, [//www.mai.liu.se/~akbj](http://www.mai.liu.se/~akbj)

"Analysis of Numerical Methods" , E. Isaacson, H. B. Keller, Dover Publications, Inc., 1994

"Numerical Methods for Scientist and Engineers", R. W. Hamming, Dover Publications, Inc., 1986

"Graphic and numerical comparison between iterative methods", Juan L. Varona, in The "Mathematical Intelligencer 24", 2002, no. 1, 37–46.

"Scientific Computing - An Introductory Survey", Michael T. Heath, in "Lecture Notes to Accompany", 2001, Second Edition

"An acceleration of the iterative processes", Gyurhan H. Nedzhibov, Milko G. Petkov, Shumen University, Bulgaria , 2002

"Lezioni - Equazioni non lineari", Lorenzo Pareschi, Università di Ferrara

"On a few Iterative Methods for Solving Nonlinear Equations", Gyurhan Nedzhibov, Laboratory of Mathematical Modelling, Shumen University, Bulgaria, 2002

"The Pegasus method for computing the root of an equation", M. Dowell and P. Jarratt, BIT 12 (1972) 503--508.

"The Java Programmer's Guide to Numerical Computation", Ronald Mak, Prentice-Hall. [//www.apropos-logic.com/nc/](http://www.apropos-logic.com/nc/)

"Numerical Analysis Topics Outline", S.-Sum Chow, 2002, [//www.math.byu.edu/~schow/work](http://www.math.byu.edu/~schow/work)

"Newton's method and high order iterations" , Xavier Gourdon , Pascal Sebah, in Number Costant and Computation , Oct. 2000, [//numbers.computation.free.fr/Constants](http://numbers.computation.free.fr/Constants)

"Radici di Equazioni Non Lineari con MathCad", A. M. Ferrari, in Appunti di LPCAC, Università di Torino

"Numerical Method - Rootfinding", in Applied Mathematics, Wolfram Reserch, [//mathworld.wolfram.com/](http://mathworld.wolfram.com/)

"Roots of Equations" , John Burkardt, School of Computational Science (SCS), Florida State University (FSU). [//www.csit.fsu.edu/~burkardt/math2070/lab_03.html](http://www.csit.fsu.edu/~burkardt/math2070/lab_03.html)

"Zoomin" a miscellanea of rootfinding subroutines in Fortran 90 by John Burkardt, 2002

"ESFR Fortran 77 Numeric Calculus Library", ESFR, 1985

"Handbook of Mathematical Functions", by M. Abramowitz and I. Stegun, Dover Publication Inc., New York.

"Zeros of Orthogonal Polynomials" , Leonardo Volpi, Foxes Team ([//digilander.libero.it/foxes/Documents](http://digilander.libero.it/foxes/Documents))

"Solutions Numeriques des Equations Algebriques", E., Durand, Tome I, Masson,Paris ,1960.

"A modified Newton method for Polynomials" , W.,Ehrlich, Comm., ACM, 1967, 10, 107-108.

- "Iteration methods for finding all the zeros of a polynomial simultaneously", O. Aberth, Math. Comp. ,1973, 27, 339-344.
- "The Ehrlich-Aberth Method for the nonsymmetric tridiagonal eigenvalue problem", D. A. Bini, L. Gemignani, F, Tisseur, AMS subject classifications. 65F15
- "Progress on the implementation of Aberth's method", D. A. Bini, G. Fiorentino, , 2002, The FRISCO Consortiumm (LTR 21.024)
- "Numerical Computation of Polynomials Roots: MPSolve v. 2" , D. A. Bini, G. Fiorentino, Università di Pisa, 2003
- "Geometries of a few single point numerical methods", Bryan McReynolds, US Military Academy, West Point, New York, 2005
- "Calculo Numérico", Neide M. B. Franco, 2002
- "Metodos Numericos" Sergio R. De Freitas, Universidade Federal de Mato Grosso do Sul, 2000
- "Appunti di Calcolo Numerico", Enrico Bertolazzi, Gianmarco Manzini, Università di Trento, C.N.R. di Pavia
- "Zeros of orthogonal polynomials", Leonardo Volpi, Foxes Team, 2003, //digilander.libero.it/foxes
- "Iterative Methods for Roots of Polynomials", Wankere R. Mekwi, University of Oxford, Trinity, 2001.
- "MultRoot - A Matlab package computing polynomial roots and multiplicities", Zhonggang Zeng, Northeastern Illinois University, 2003
- "Simultaneous Determination of all the zeros of a linear combination of legendre polynomial", Nazir Ahmad Mir*, Aman Ullah Khan and Zahida Akram, Bahauddin Zakariya University, Multan , Pakistan.
- "PRECISE: Efficient Multiprecision Evaluation of Algebraic Roots and Predicates for Reliable Geometric Computation", Shankar Krishnan, AT&T Labs Research, Mark Foskey, Univ. of North Carolina, Tim Culver, John Keyser, Texas A&M Univ., Dinesh Manocha, Univ. of North Carolina
- "Durand Kerner Root-Finding Method for Generalized Tridiagonal Eigenproblem", Kuiyuan Li, Department of Mathematics and Statistics, University of West Florida, Pensacola

Allegati

Risultati dei casi di test

Costo per metodo e test

Method / Test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bisect	350	350	336	364	350	119	126	350	343	287	371	357	350	343	175
Pegasus	108	144	96	108	120	576	552	84	120	168	192	180	120	156	516
Regula Falsi	360	1030	780	160	90	?	?	60	90	?	?	1750	420	2320	?
Secant	140	?	90	80	?	360	320	?	?	?	100	?	160	?	350
Brent	270	300	270	360	270	1050	990	210	300	330	240	330	360	300	1080
Muller	195	273	234	312	312	702	975	234	351	585	?	1053	1677	273	351
Newton	72	84	84	?	?	324	288	?	?	108	132	96	564	96	300
Rheinboldt 2	231	1089	1155	1287	264	759	594	198	297	363	462	396	396	330	957
Secant back	120	288	108	108	108	432	384	72	108	516	132	612	132	444	420
Star E 21	75	105	90	135	285	345	375	105	?	120	135	?	1170	105	345
Steffenson	154	?	?	84	?	420	350	?	?	84	?	?	378	?	364
Halley	88	88	66	?	?	352	308	88	88	132	110	110	286	88	374
Halley FD	102	119	102	119	?	374	204	119	221	119	119	221	255	119	323
Parabola	135	189	135	243	216	702	459	189	297	216	405	405	891	189	243
Cheby FD	144	234	90	270	?	522	558	?	?	360	144	?	414	?	522
Parabola inv.	140	200	100	?	?	400	620	280	?	580	180	?	480	220	560
Fraction	108	126	90	180	162	?	324	144	414	234	162	162	684	126	342

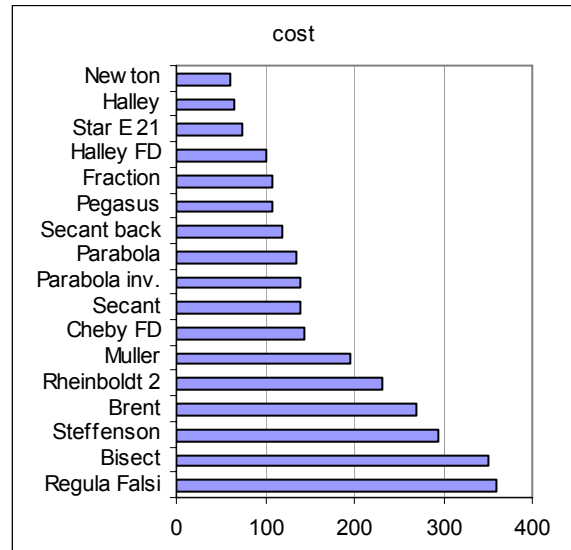
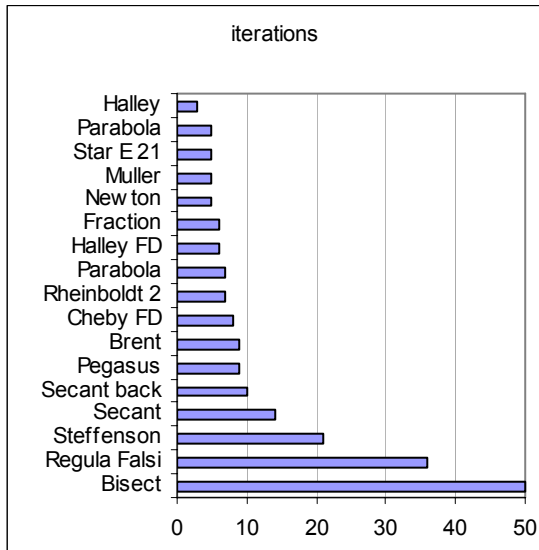
? = obiettivo non raggiunto

Numero d'iterazioni per metodo e test

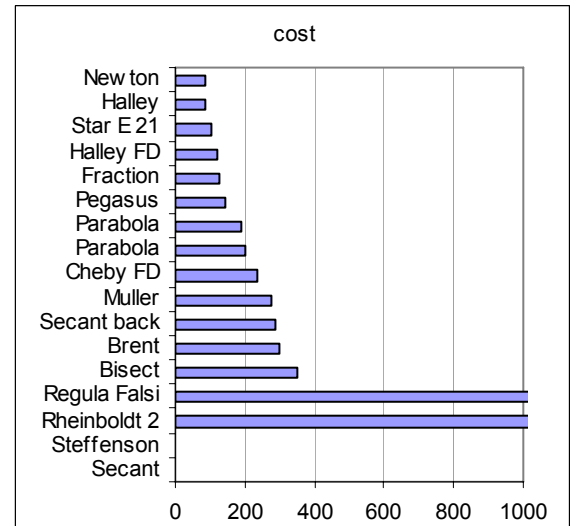
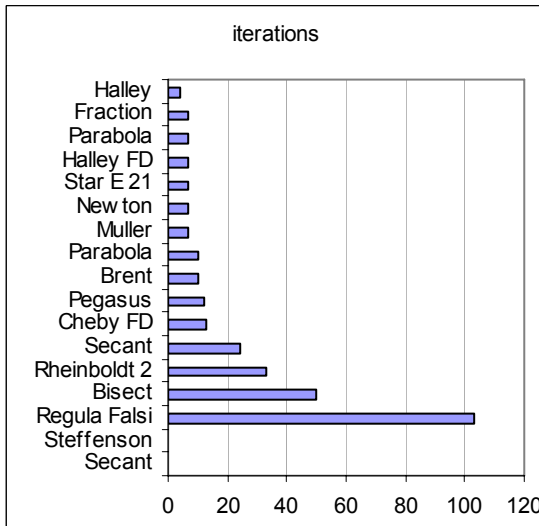
Method / Test	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bisect	50	50	48	52	50	17	18	50	49	41	53	51	50	49	25
Pegasus	9	12	8	9	10	48	46	7	10	14	16	15	10	13	43
Regula Falsi	36	103	78	16	9	?	?	6	9	?	?	175	42	232	?
Secant	14	?	9	8	?	36	32	?	?	?	10	?	16	?	35
Brent	9	10	9	12	9	35	33	7	10	11	8	11	12	10	36
Muller	5	7	6	8	8	18	25	6	9	15	?	27	43	7	9
Newton	6	7	7	?	?	27	24	?	?	9	11	8	47	8	25
Rheinboldt 2	7	33	35	39	8	23	18	6	9	11	14	12	12	10	29
Secant back	10	24	9	9	9	36	32	6	9	43	11	51	11	37	35
Star E 21	5	7	6	9	19	23	25	7	?	8	9	?	78	7	23
Steffenson	11	?	?	6	?	30	25	?	?	6	?	?	27	?	26
Halley	4	4	3	?	?	16	14	4	4	6	5	5	13	4	17
Halley FD	6	7	6	7	?	22	12	7	13	7	7	13	15	7	19
Parabola	5	7	5	9	8	26	17	7	11	8	15	15	33	7	9
Cheby FD	8	13	5	15	?	29	31	?	?	20	8	?	23	?	29
Parabola inv.	7	10	5	?	?	20	31	14	?	29	9	?	24	11	28
Fraction	6	7	5	10	9	?	18	8	23	13	9	9	38	7	19

? = obiettivo non raggiunto

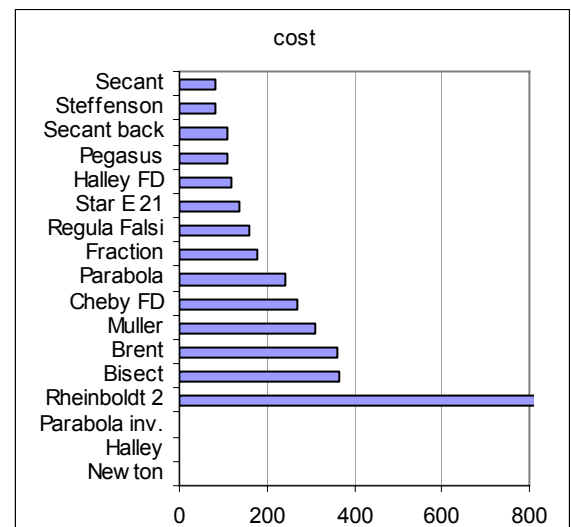
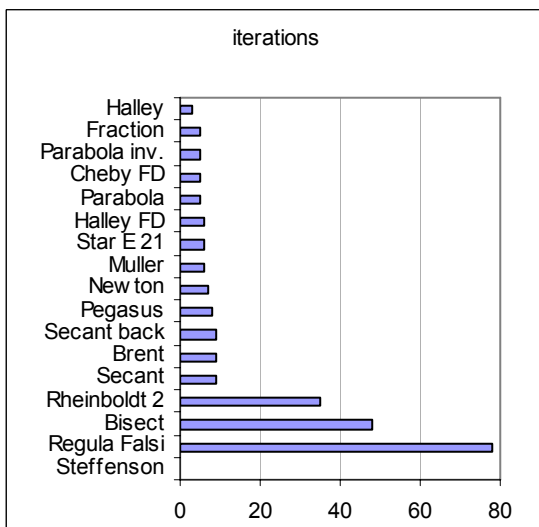
Test 01



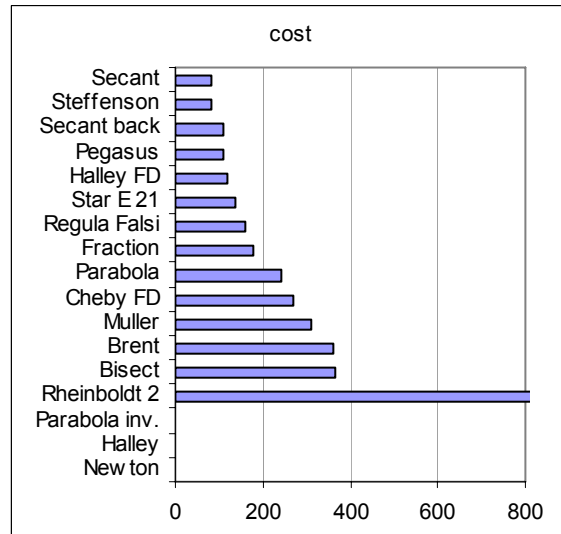
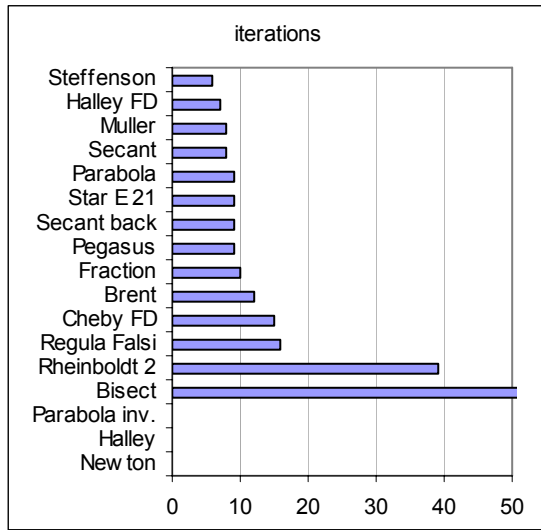
Test 02



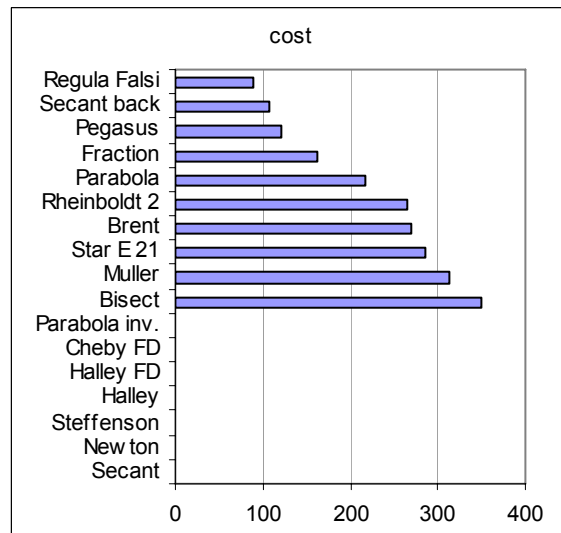
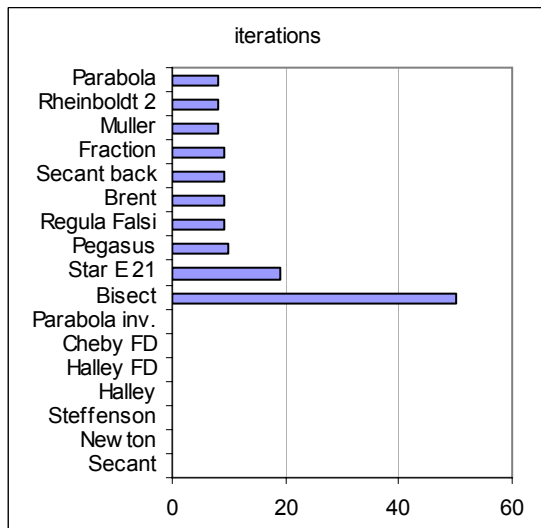
Test 3



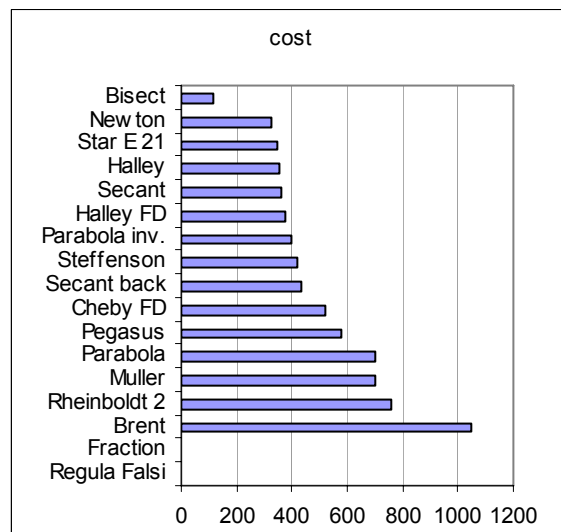
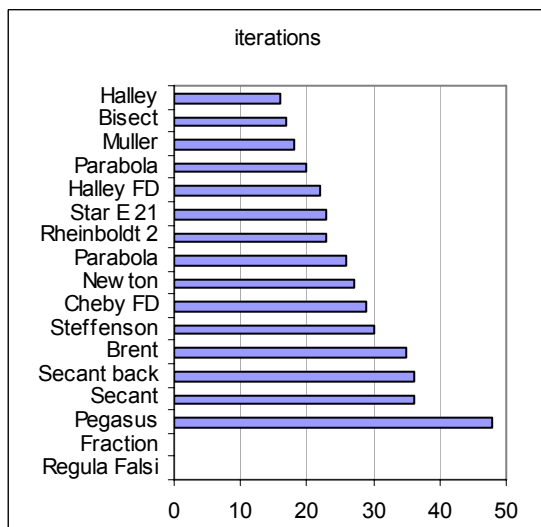
Test 4



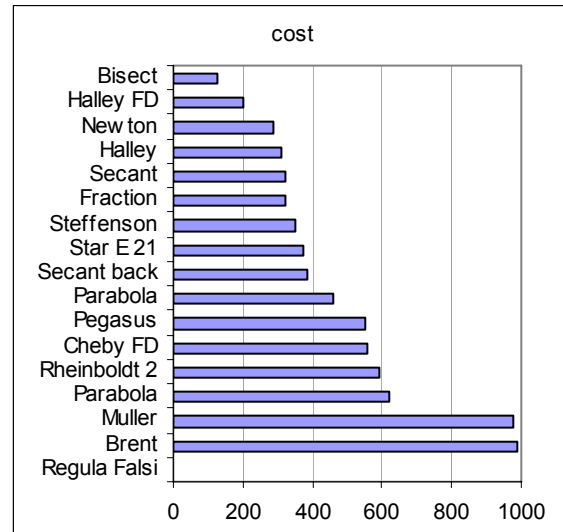
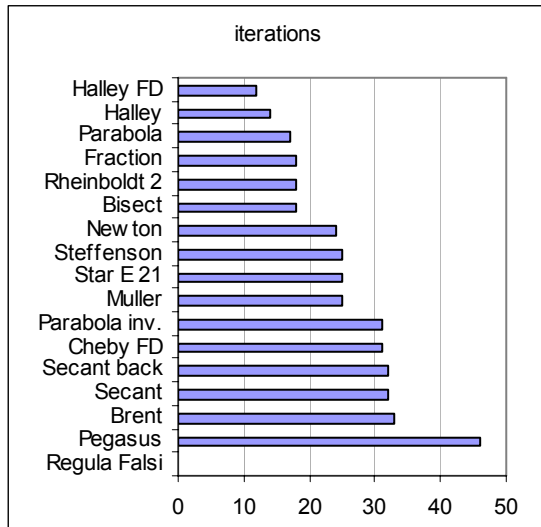
Test 5



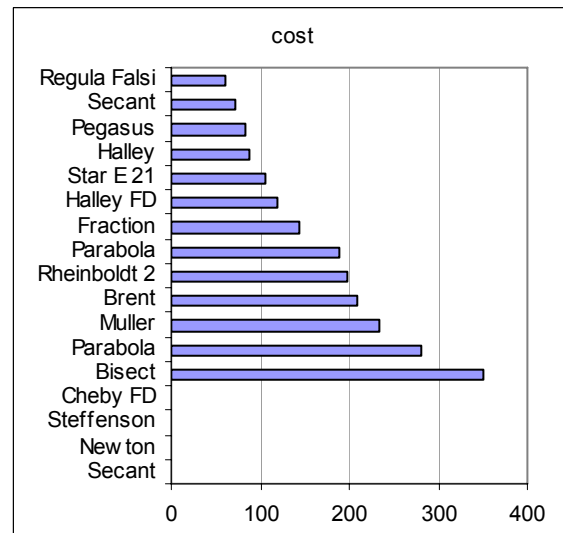
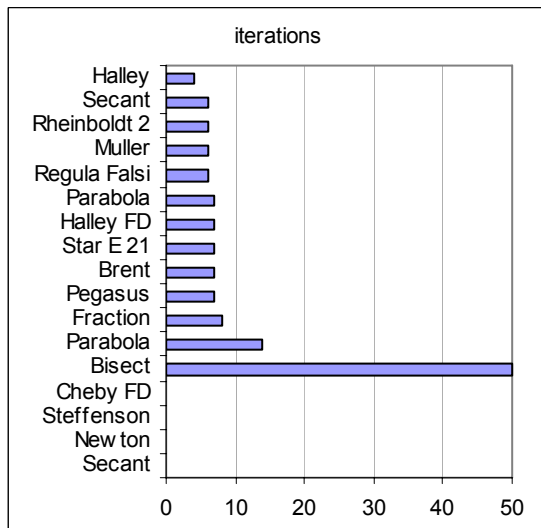
Test 6



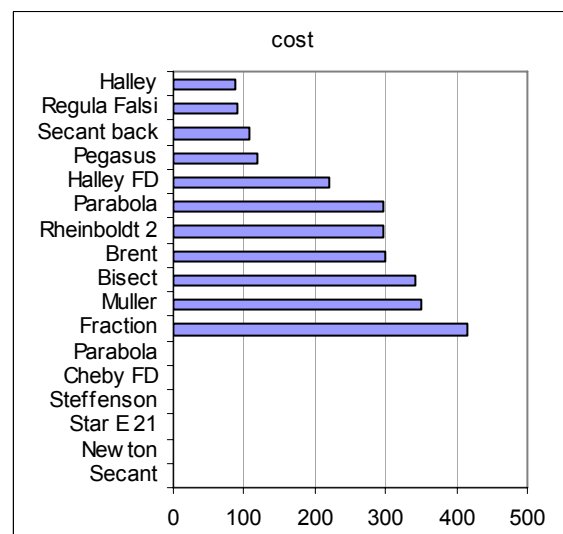
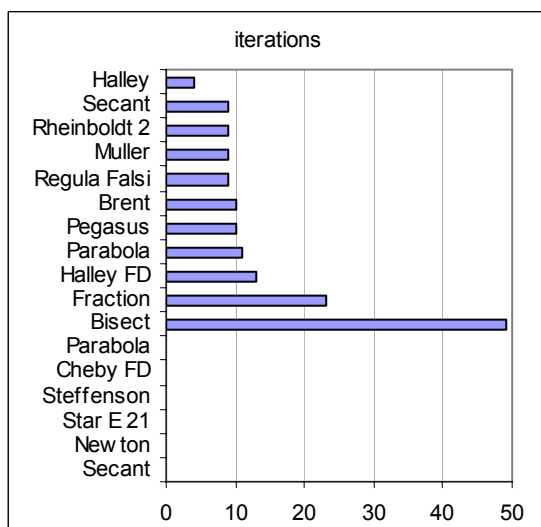
Test 7



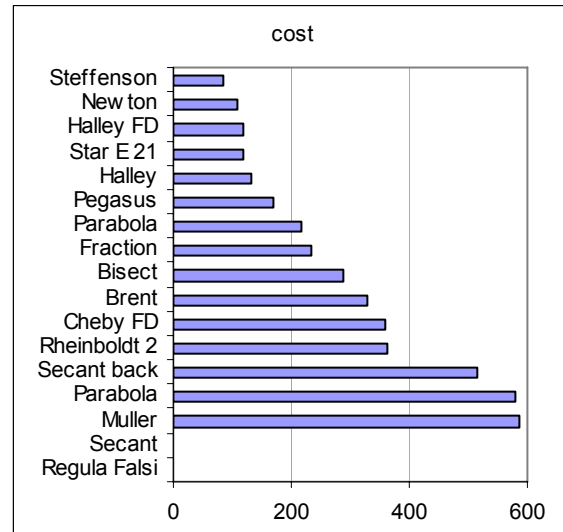
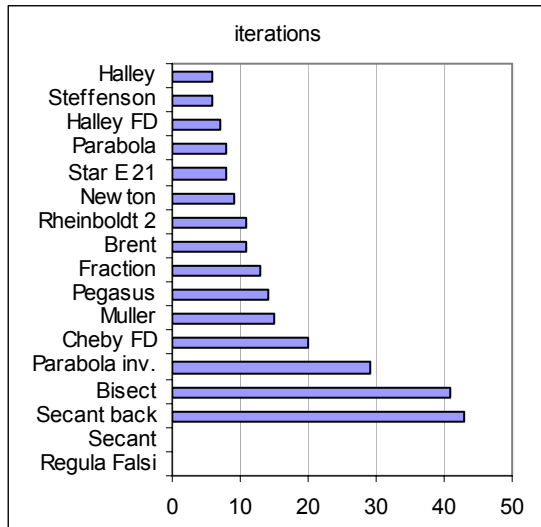
Test 8



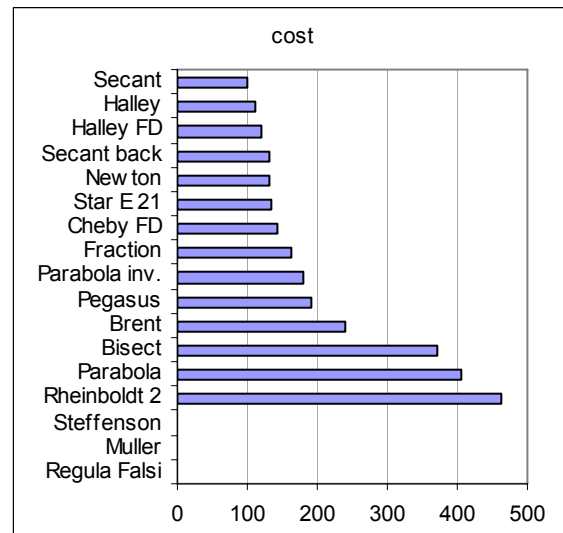
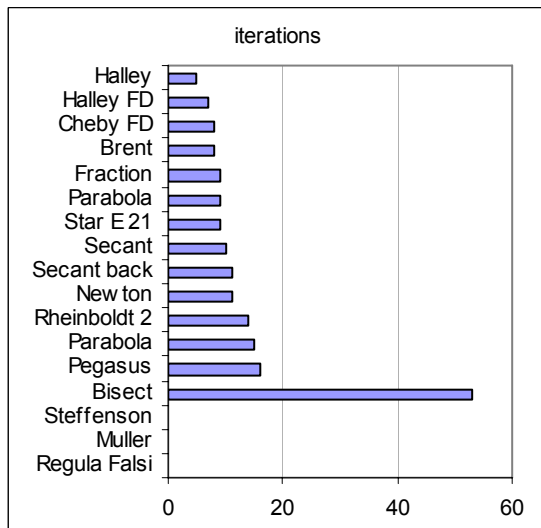
Test 9



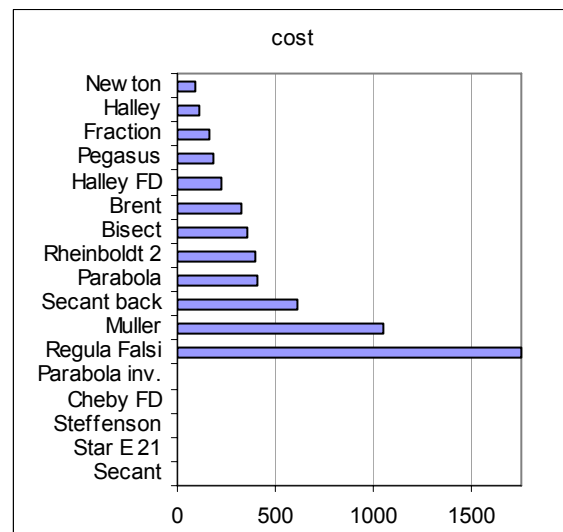
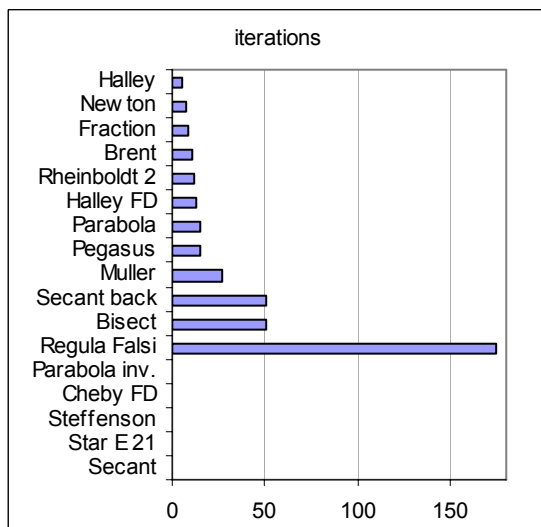
Test 10



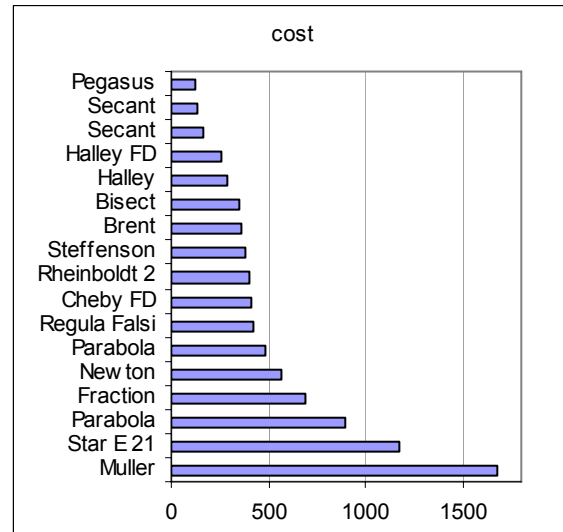
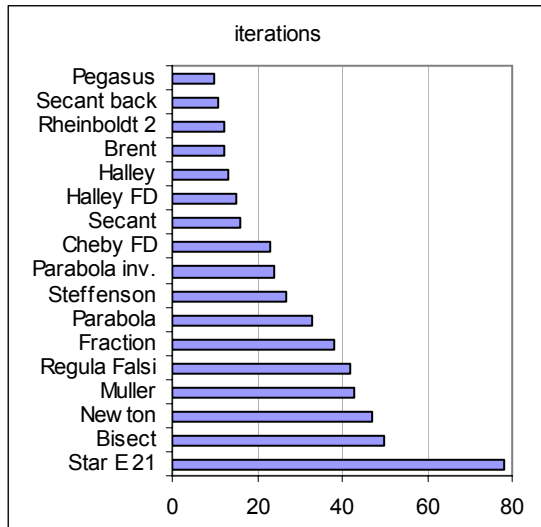
Test 11



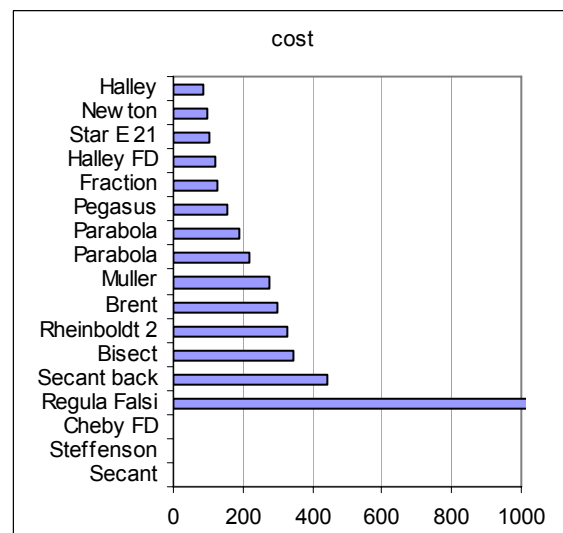
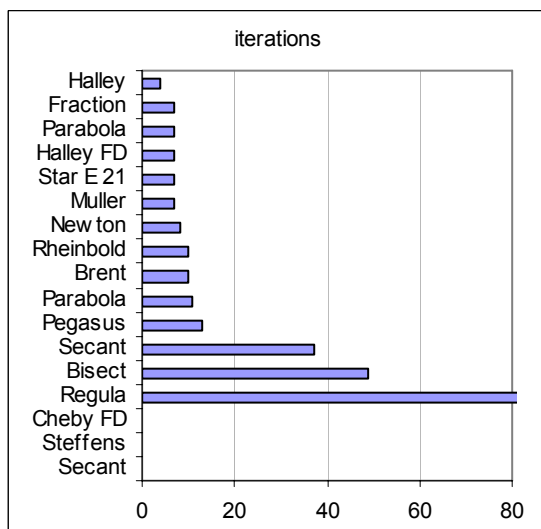
Test 12



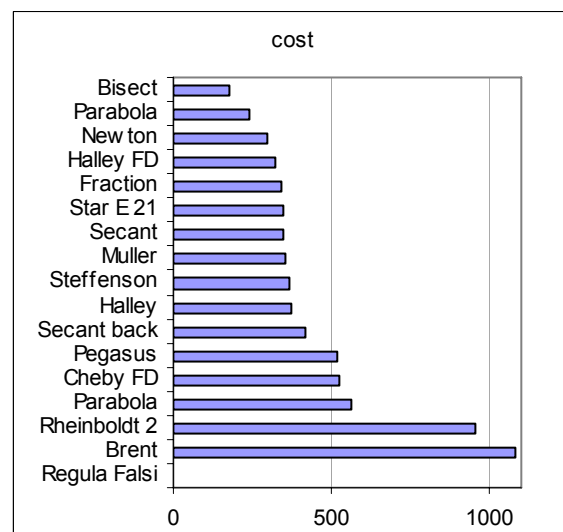
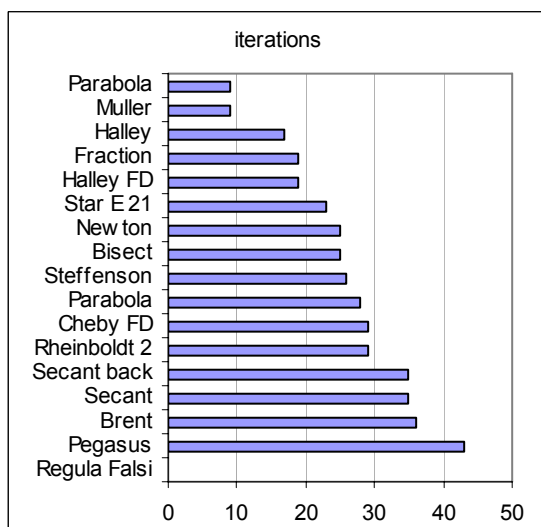
Test 13



Test 14



Test 15



Potenze del binomio complesso (x+iy)

Riportiamo lo sviluppo delle prime 9 potenze del binomio complesso (x + i·y)

$$(x + i \cdot y)^2 = x^2 - y^2 + 2 \cdot i \cdot x \cdot y;$$

$$(x + i \cdot y)^3 = x \cdot (x^2 - 3 \cdot y^2) + i \cdot y \cdot (3 \cdot x^2 - y^2);$$

$$(x + i \cdot y)^4 = x^4 - 6 \cdot x^2 \cdot y^2 + y^4 + 4 \cdot i \cdot x \cdot y \cdot (x^2 - y^2);$$

$$(x + i \cdot y)^5 = x \cdot (x^4 - 10 \cdot x^2 \cdot y^2 + 5 \cdot y^4) + i \cdot y \cdot (5 \cdot x^4 - 10 \cdot x^2 \cdot y^2 + y^4);$$

$$(x + i \cdot y)^6 = (x^2 - y^2) \cdot (x^4 - 14 \cdot x^2 \cdot y^2 + y^4) + 2 \cdot i \cdot x \cdot y \cdot (3 \cdot x^4 - 10 \cdot x^2 \cdot y^2 + 3 \cdot y^4);$$

$$(x + i \cdot y)^7 = x \cdot (x^6 - 21 \cdot x^4 \cdot y^2 + 35 \cdot x^2 \cdot y^4 - 7 \cdot y^6) + i \cdot y \cdot (7 \cdot x^6 - 35 \cdot x^4 \cdot y^2 + 21 \cdot x^2 \cdot y^4 - y^6);$$

$$(x + i \cdot y)^8 = x^8 - 28 \cdot x^6 \cdot y^2 + 70 \cdot x^4 \cdot y^4 - 28 \cdot x^2 \cdot y^6 + y^8 + 8 \cdot i \cdot x \cdot y \cdot (x^2 - y^2) \cdot (x^4 - 6 \cdot x^2 \cdot y^2 + y^4);$$

$$(x + i \cdot y)^9 = x \cdot (x^8 - 36 \cdot x^6 \cdot y^2 + 126 \cdot x^4 \cdot y^4 - 84 \cdot x^2 \cdot y^6 + 9 \cdot y^8) + i \cdot y \cdot (9 \cdot x^8 - 84 \cdot x^6 \cdot y^2 + 126 \cdot x^4 \cdot y^4 - 36 \cdot x^2 \cdot y^6 + y^8)$$

Software

Il codice usato per lo studio delle routine zerofinder è stato costruito sul modello del software "Zoomin - a miscellanea of rootfinding subroutines" sviluppato in Fortran 90 da John Burkardt della Florida State University (FSU).

Gran parte delle routine sono state tradotte e adattate in Visual Basic da questo ottimo lavoro

La routine Pegaso è stata tradotta e adattata dalla libreria Fortran 77 ESFR.

Per usare le routine basta semplicemente selezionare il testo seguente e importarlo in un modulo VB

```

*****
'
' Zerofinder Miscellanea in VB
'
' Arranged by Leonardo Volpi, Foxes Team, Oct. 2005
'
' Credits
' The original code ZOOMIN_f90 was written in Fortran 90 by John Burkardt
' and derived from the original code of Harold Deiss in BASIC
' The original Pegaso routine was written by Gisela E. Muellges for the ESFR Fortran
' Library
'
'-----
' Methods without derivatives
'-----
' PEGASUS      implement the Dowell-Jarrat method (a modified regula falsi method)
' BISECT      carries out the bisection method.
' REGULA      implements the Regula Falsi method.
' BRENT       implements the Wijngaarden-Dekker-Brent zero finder.
' MULLER      implements Muller's method.
' RHEIN2      implements the Rheinboldt method.
' SECANT      implements the secant method.
' CHEBY_FD    carries out the extended secant algorithm.
' STAR_E21    implements the Traub *E21 method.
' STEFFENSON  implements Steffenson's method.
' HALLEY_FD   implements Halley's method, with finite differences.
' PARABOLA    implements the vertical parabola interpolation method.
' PARABOLA_INV implements the inverse parabola interpolation method.
' SECANT_BACK implements the safe secant method with back-step
' FRACTION    implements the linear fraction interpolation method.
'-----
'
' Methods with 1st derivative
'-----
' NEWTON      implements Newton's method.
'-----
'
' Methods with 2nd derivative
'-----
' HALLEY1     implements Halley's method.
'-----
'
' All the routines accepts these parameters
'
'   Input, real ABSEERR, an error tolerance.
'
'   Output, integer IERROR, error indicator.
'   0, no error occurred.
'   nonzero, an error occurred, and the iteration was halted.
'
'   Output, integer K, the number of steps taken.
'
'   Input, integer KMAX, the maximum number of iterations allowed.
'
*****
Option Explicit
*****
'
' Evaluation function or its derivatives
'

```

```

' x is the point where you want to evaluate
' d switches the calculus:
'   d = 0 returns f(x)
'   d = 1 returns f'(x)
'   d = 2 returns f''(x)
'
'*****

Function Funct(x, Optional d = 0)
Dim y As Double
  Select Case d
    Case 0
      'write here the funcyion
      y = (3 * x / 2) ^ 3 - 1
    Case 1
      'write here its 1st derivative
      y = 81 / 8 * x ^ 2
    Case 2
      'write here its 2nd derivative
      y = 81 / 4 * x
  End Select
  Funct = y 'return the value
End Function

'-- Auxiliary functions --

Private Function min(x, y)
  If x < y Then min = x Else min = y
End Function

Private Sub r_swap(x, y)
'exchange two variables
Dim tmp#
  tmp = x
  x = y
  y = tmp
End Sub

Private Function sign(x, y)
'simulate the Fortran (silly!) sign function
  If y >= 0 Then
    sign = Abs(x)
  Else
    sign = -Abs(x)
  End If
End Function

'-- End of Auxiliary functions --

'*****
'
' BISECT carries out the bisection method.
'
' Parameters:
'   Input/output, real X, X1, two points defining the interval in which the
'   search will take place. F(X) and F(X1) should have opposite signs.
'   On output, X is the best estimate for the root, and X1 is
'   a recently computed point such that F changes sign in the interval
'   between X and X1.
'*****

Sub bisect(x, x1, abserr, ierror, k, kmax)
  Dim dx#
  Dim fx#, fx1#, fx2#, x2#
'
' Initialization.
'
  ierror = 0
  k = 0
'
' Evaluate the function at the starting points.
'
  fx = Funct(x, 0)
  fx1 = Funct(x1, 0)
'
' Set XPOS and XNEG to the X values for which F(X) is positive
' and negative, respectively.

```

```

,
  If (fx >= 0# And fx1 <= 0#) Then

  ElseIf (fx <= 0# And fx1 >= 0#) Then
    Call r_swap(x, x1)
    Call r_swap(fx, fx1)
  Else
    ierror = 1
    Exit Sub
  End If
,
  Iteration loop:
,
  Do
,
  If the error tolerance is satisfied, then exit.
,
  If (Abs(x - x1) <= abserr) Then
    Exit Sub
  End If

  If (Abs(fx) <= abserr) Then
    Exit Sub
  End If

  k = k + 1

  If (k > kmax) Then
    ierror = 2
    Exit Sub
  End If
,
  Update the iterate and function values.
,
  x2 = (x1 + x) / 2
  fx2 = Funct(x2, 0)

  If (fx2 >= 0) Then
    x = x2
    fx = fx2
  Else
    x1 = x2
    fx1 = fx2
  End If

  Loop
End Sub

*****
Zero_Pegaso determines the zero by applying the Pegasus-method
,
Parameters:
a,b : endpoints of the interval containing a zero.
MaxIt : maximum number of iteration steps.
xsi : zero or approximate value for the zero
      of the function Funct.
x1,x2 : the last two iterates, so that [X1,X2] contains a
        zero of Funct.
Numit : number of iteration steps executed.
Ierr : =-2, ABSERR is negative or both equal zero,
        or MAXIT < 1.
        =-1, the condition Funct(a)*Funct(b) < 0.0 is not met.
        = 0, a or b already are a zero of Funct.
        = 1, XSI is a zero with ABS(Func(XSI)) < 4* machine
        constant.
        = 2, break-off criterion has been met, XSI:=X2 if
        ABS(Funct(X2)) <= ABS(Funct(X1)),
        otherwise XSI:=X1.
        The absolute error of the computed zero is less
        than or equal to ABS(X1-X2).
        = 3, the maximum number of iteration steps was
        reached without meeting the break-off criterion.
abserr : absolute error tolerance
,
Derived from the original FORTRAN version by
author : Gisela Engeln-Muellges

```

```

' date      : 09.02.1985
' source    : ESFR FORTRAN 77 Library
'
'*****
Sub Zero_Pegaso(a, b, xsi, x1, x2, NumIt, MaxIt, Ierr, abserr)
Dim F1, F2, F3, i, x3, s12, tol
  NumIt = 0
  tol = 4 * abserr
  x1 = a
  x2 = b
' calculating the functional values at the endpoints A and B.
  F1 = Funct(x1)
  F2 = Funct(x2)
' testing for alternate signs of Funct at A and B: Funct(A)*Funct(B) < 0.0 .
  If (F1 * F2 > 0) Then
    Ierr = -1
    Exit Sub
  ElseIf (F1 * F2 = 0) Then
    Ierr = 0
    Exit Sub
  End If
'
' executing the Pegasus-method.
  For i = 1 To MaxIt
    NumIt = i
' testing whether the value of F2 is less than four times
    If (Abs(F2) < tol) Then
      xsi = x2
      Ierr = 1
      Exit Sub
' testing for the break-off criterion.
    ElseIf (Abs(x2 - x1) <= Abs(x2) * abserr) Then
      xsi = x2
      If (Abs(F1) < Abs(F2)) Then xsi = x1
      Ierr = 2
      Exit Sub
    Else
' calculating the secant slope.
      s12 = (F2 - F1) / (x2 - x1)
' calculating the secant intercept X3 with the x-axis.
      x3 = x2 - F2 / s12
' calculating a new functional value at X3.
      F3 = Funct(x3)
' definition of the endpoints of a smaller inclusion interval.
      If (F2 * F3 <= 0) Then
        x1 = x2
        F1 = F2
      Else
        F1 = F1 * F2 / (F2 + F3)
      End If
      x2 = x3
      F2 = F3
    End If
  Next i
  Ierr = 3
End Sub

'*****
'
' REGULA implements the Regula Falsi method.
'
' Parameters:
'
' Input/output, real X, X1.
' On input, two distinct points that start the method.
' On output, X is an approximation to a root of the equation
' which satisfies abs ( F(X) ) < ABSERR, and X1 is the previous
' estimate.
'
'*****

Sub regula(x, x1, abserr, ierror, k, kmax)
Dim fx, fx1, dx, x2, fx2
' Initialization.
'
  ierror = 0
  k = 0

```

```

fx = Funct(x, 0)
fx1 = Funct(x1, 0)

If (fx < 0#) Then
  Call r_swap(x, x1)
  Call r_swap(fx, fx1)
End If

dx = 1
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
  If (Abs(fx) <= abserr) Then
    Exit Sub
  End If
  If (Abs(dx) <= abserr) Then
    Exit Sub
  End If

  k = k + 1

  If (k > kmax) Then
    ierror = 2
    Exit Sub
  End If
'
' Set the increment.
'
  dx = -fx1 * (x - x1) / (fx - fx1)
'
' Update the iterate and function values.
'
  x2 = x1 + dx
  fx2 = Funct(x2, 0)

  If (fx2 >= 0#) Then
    x = x2
    fx = fx2
  Else
    x1 = x2
    fx1 = fx2
  End If

Loop

End Sub

'*****
'
' SECANT implements the secant method.
'
' Parameters:
'
'   Input/output, real X, X1.
'   On input, two distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 is the previous
'   estimate.
'
' Reference:
'
'   Joseph Traub,
'   Iterative Methods for the Solution of Equations,
'   Prentice Hall, 1964.
'   Input, integer KMAX, the maximum number of iterations allowed.
'*****

Sub secant(x, x1, abserr, ierror, k, kmax)

Dim fx, fx1, dx
'

```

```

' Initialization.
'
ierror = 0
k = 0
fx = Funct(x, 0)
fx1 = Funct(x1, 0)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
  If (Abs(fx) <= abserr) Then
    Exit Sub
  End If

  k = k + 1

  If (k > kmax) Then
    ierror = 2
    Exit Sub
  End If

  If ((fx - fx1) = 0#) Then
    ierror = 3
    Exit Sub
  End If
'
' Set the increment.
'
  dx = -fx * (x - x1) / (fx - fx1)
'
' Remember current data for next step.
'
  x1 = x
  fx1 = fx
'
' Update the iterate and function values.
'
  x = x + dx
  fx = Funct(x, 0)

Loop

End Sub

'*****
'
' BRENT implements the Wijngaarden-Dekker-Brent zero finder.
'
' Parameters:
'
'   Input/output, real X, X1. On input, two points defining the interval
'   in which the search will take place. F(X) and F(X1) should have opposite
'   signs. On output, X is the best estimate for the root, and X1 is
'   a recently computed point such that F changes sign in the interval
'   between X and X1.
'
' Reference:
'
'   Richard Brent,
'   Algorithms for Minimization without Derivatives,
'   Prentice Hall, 1973.
'*****

Sub brent(x, x1, abserr, ierror, k, kmax)
'
Dim fx, fx1, x2, fx2, d, e, em, tol, s, p, q, r, dx, t
'
' Initialization.
'
ierror = 0
k = 0
fx = Funct(x, 0)
fx1 = Funct(x1, 0)
x2 = x1

```

```

fx2 = fx1
tol = abserr
,
Iteration loop:
,
Do
  k = k + 1

  If (k > kmax) Then
    ierror = 2
    Exit Sub
  End If

  If (fx1 > 0 And fx2 > 0) Or (fx1 < 0 And fx2 < 0) Then
    x2 = x
    fx2 = fx
    d = x1 - x
    e = x1 - x
  End If

  If (Abs(fx2) < Abs(fx1)) Then
    Call r_swap(x1, x2)
    Call r_swap(fx1, fx2)
  End If

  em = 0.5 * (x2 - x1)

  If (Abs(em) <= tol) Or (Abs(fx1) < tol) Then
    x = x1
    Exit Sub
  End If

  If (Abs(e) < tol) Or (Abs(fx) <= Abs(fx1)) Then

    d = em
    e = em

  Else

    s = fx1 / fx

    If (x = x2) Then
      p = 2 * em * s
      q = 1 - s
    Else
      t = fx / fx2
      r = fx1 / fx2
      p = s * (2 * em * t * (t - r) - (x1 - x) * (r - 1))
      q = (t - 1) * (r - 1) * (s - 1)
    End If

    If (p > 0) Then
      q = -q
    Else
      p = -p
    End If

    s = e
    e = d

    If ((2 * p < 3 * em * q - Abs(tol * q)) Or (p < Abs(0.5 * s * q))) Then
      d = p / q      'Accept interpolation.
    Else
      d = em        'Interpolation failed, use bisection.
      e = em
    End If

  End If
,
Set the increment.
,
  If (Abs(d) > tol) Then
    dx = d
  ElseIf (em > 0) Then
    dx = tol
  Else
    dx = -tol

```

```

    End If
,
' Remember current data for next step.
,
    x = x1
    fx = fx1
,
' Update the iterate and function values.
,
    x1 = x1 + dx
    fx1 = Funct(x1, 0)

Loop
End Sub

*****
' MULLER implements Muller's method. (metodo della parabola)
,
' Parameters:
,
'   Input/output, real X, X1, X2.
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'   previous estimates.
,
*****

Sub muller(x, x1, x2, abserr, ierror, k, kmax)
,
Dim fx1, fx2, fx, q, a, b, c, term, dx
,
' Initialization.
,
    ierror = 0
    k = 0
    x = (x1 + x2) / 2
    fx2 = Funct(x2, 0)
    fx1 = Funct(x1, 0)
    fx = Funct(x, 0)
,
' Iteration loop:
,
Do
,
' If the error tolerance is satisfied, then exit.
,
    If (Abs(fx) <= abserr) Then
        Exit Sub
    End If

    k = k + 1

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If

    q = (x - x1) / (x1 - x2) 'normalized variable    0 < q < 1

    a = q * fx - q * (1 + q) * fx1 + q ^ 2 * fx2
    b = (2 * q + 1) * fx - (1 + q) ^ 2 * fx1 + q ^ 2 * fx2
    c = (1 + q) * fx

    term = b ^ 2 - 4 * a * c
    If term < 0 Then term = 0
    term = Sqr(term)
    If (b < 0) Then
        term = -term
    End If
,
' Set the increment.
,
    dx = -(x - x1) * 2 * c / (b + term)
,
' Remember current data for next step.

```

```

'
'   x2 = x1
'   fx2 = fx1
'   x1 = x
'   fx1 = fx
'
'   Update the iterate and function values.
'
'   x = x + dx
'   fx = Funct(x, 0)
Loop
End Sub

'*****
'
' parabola implements the vertical parabola interpolation method.
'
' Parameters:
'
'   Input/output, real X, X1, X2
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'   previous estimates.
'
' Author: Leonardo Volpi
'
'*****

Sub parabola(x, x1, x2, abserr, ierror, k, kmax)
Dim y0, y1, y2, x0, a, b, x10, x12, x02, d1, d0, delta, dx
'
' Initialization.
'
' ierror = 0
' k = 0
' x0 = x1
' x1 = x2
' x2 = (x1 + x0) / 2
' y2 = Funct(x2, 0)
' y1 = Funct(x1, 0)
' y0 = Funct(x0, 0)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
' If (Abs(y2) <= abserr) Then
'   Exit Sub
' End If
' If (Abs(x2 - x1) <= abserr) Then
'   Exit Sub
' End If
' k = k + 1
'
' If (k > kmax) Then
'   ierror = 2
'   Exit Sub
' End If
'
' parabolas coefficients a, b a*(x-x2)^2+b*(x-x2)+y2
' x10 = x1 - x0
' x12 = x1 - x2
' x02 = x0 - x2
' d1 = (y1 - y2) / x12
' d0 = (y0 - y2) / x02
' a = (d1 - d0) / x10
' b = (x12 * d0 - x02 * d1) / x10
' delta = b ^ 2 - 4 * a * y2
' If delta < 0 Then delta = 0
' dx = -2 * y2 / (b + Sgn(b) * Sqr(delta))
'
' Remember current data for next step.

```

```

    x0 = x1: y0 = y1
    x1 = x2: y1 = y2
,
' Update the iterate and function values.
,
    x2 = x2 + dx
    y2 = Funct(x2, 0)
Loop
    x = x2
End Sub

*****
'
' RHEIN2 implements the Rheinboldt bisection - secant - inverse quadratic method.
'
' Parameters:
'
'   Input/output, real X, X1.
'   On input, two distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 is the previous
'   estimate.
'
' Reference:
'
'   W C Rheinboldt,
'   Algorithms for Finding Zeros of a Function,
'   UMAP Journal,
'   Volume 2, 1, 1981, pages 43 - 72.
'
*****

Sub rhein2(x, x1, abserr, ierror, k, kmax)

    Dim dx, em, forced, fx, fx1, fx2, i, ps
    Dim piq, qiq, qs, stpmin, t, u, v, w, x2
,
' Initialization.
,
    i = 0
    ierror = 0
    k = 0
    fx = Funct(x, 0)
    fx1 = Funct(x1, 0)

    If (Abs(fx) > Abs(fx1)) Then
        Call r_swap(x, x1)
        Call r_swap(fx, fx1)
    End If

    x2 = x1
    fx2 = fx1
    k = 0
    t = 0.5 * Abs(x - x1)

,
' Iteration loop:
,
Do
,
' If the error tolerance is satisfied, then exit.
,
    If (Abs(fx) <= 5 * abserr) Then
        Exit Sub
    End If

    k = k + 1

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If

    i = i + 1

    em = 0.5 * (x1 - x)

,
' Compute the numerator and denominator for secant step.
,
```

```

If (2 * Abs(x2 - x) < Abs(x1 - x)) Then
  ps = (x - x2) * fx
  qs = fx2 - fx
Else
  ps = (x - x1) * fx
  qs = fx1 - fx
End If

If (ps < 0) Then
  ps = -ps
  qs = -qs
End If
,
Compute the numerator and denominator for inverse quadratic.
,
piq = 0
qiq = 0

If (x1 <> x2) Then
  u = fx / fx2
  v = fx2 / fx1
  w = fx / fx1
  piq = u * (2 * em * v * (v - w) - (x - x2) * (w - 1))
  qiq = (u - 1) * (v - 1) * (w - 1)

  If (piq > 0#) Then
    qiq = -qiq
  End If

  piq = Abs(piq)
End If
,
Save the old minimum residual point.
,
x2 = x
fx2 = fx

stpmin = (Abs(x) + Abs(em) + 1) * abserr
,
Choose bisection, secant or inverse quadratic step.
,
forced = False

If (i > 3) Then
  If (8 * Abs(em) > t) Then
    forced = True
  Else
    i = 0
    t = em
  End If
End If
,
Set the increment.
,
If (forced) Then
  dx = em
ElseIf (piq < 1.5 * em * qiq) And (Abs(piq) > Abs(qiq) * stpmin) Then
  dx = piq / qiq
ElseIf (ps < qs * em) And (Abs(ps) > Abs(qs) * stpmin) Then
  dx = ps / qs
Else
  dx = em
End If
,
Update the iterate and function values.
,
x = x + dx
fx = Funct(x, 0)
,
Set the new X1 as either X1 or X2, depending on whether
F(X1) or F(X2) has the opposite sign from F(X).
,
If (sign(1, fx) = sign(1, fx1)) Then
  x1 = x2
  fx1 = fx2
End If
,
Force ABS ( FX ) <= ABS ( FX1 ).

```

```

    If (Abs(fx) > Abs(fx1)) Then
        Call r_swap(x, x1)
        Call r_swap(fx, fx1)
    End If

Loop

End Sub

'*****
'
' CHEBY_FD carries out the Chebyshev-Householder method with finite differences.
'
' Parameters:
'
'   Input/output, real X, X1, X2.
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are two previous
'   estimates.
'*****

Sub cheby_fd(x, x1, x2, abserr, ierror, k, kmax)
    Dim d1, d2, dx, fx, fx1, fx2
    ' Initialization.
    ierror = 0
    k = 0

    x = (x1 + x2) / 2

    fx1 = Funct(x1, 0)
    fx2 = Funct(x2, 0)
    d1 = (fx1 - fx2) / (x1 - x2)

    If (d1 = 0) Then
        ierror = 3
        Exit Sub
    End If

    fx = Funct(x, 0)

    ' Iteration loop:
    Do
        ' If the error tolerance is satisfied, then exit.

        If (Abs(fx) <= abserr) Then
            Exit Sub
        End If

        k = k + 1

        If (k > kmax) Then
            ierror = 2
            Return
        End If

        d2 = d1

        If (x = x1) Then
            ierror = 3
            Exit Sub
        End If

        d1 = (fx - fx1) / (x - x1)

        If (d1 = 0) Then
            ierror = 3
            Exit Sub
        End If

        If (fx2 = fx1) Then
            ierror = 3

```

```

        Exit Sub
    End If
'
' Set the increment.
'
    dx = -fx / d1 + (fx * fx1 / (fx - fx2)) * (1 / d1 - 1 / d2)
'
' Remember current data for next step.
'
    x2 = x1
    fx2 = fx1
    x1 = x
    fx1 = fx
'
' Update the iterate and function values.
'
    x = x + dx
    fx = Funct(x, 0)

Loop
End Sub

*****
' STAR_E21 implements the Traub *E21 method.
'
' Parameters:
'
'   Input/output, real X, X1, X2.
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'   previous estimates.
'
' Reference:
'
'   Joseph Traub,
'   Iterative Methods for the Solution of Equations,
'   Prentice Hall, 1964, page 234.
'
*****

Sub star_e21(x, x1, x2, abserr, ierror, k, kmax)
' Initialization.
Dim fx, fx1, fx2, d, d1, d2, m, dx
'
    ierror = 0
    k = 0
    x = (x1 + x2) / 2
    fx = Funct(x, 0)
    fx1 = Funct(x1, 0)
    fx2 = Funct(x2, 0)
    d1 = (fx1 - fx2) / (x1 - x2)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
    If (Abs(fx) <= abserr) Then
        Exit Sub
    End If

    k = k + 1

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If

    d2 = d1

    If (x = x1) Then
        ierror = 3
        Exit Sub
    End If

```

```

    d1 = (fx - fx1) / (x - x1)

    If (x = x2) Then
        ierror = 3
        Exit Sub
    End If

    d = (fx - fx2) / (x - x2)
    m = d1 + d - d2

    If (m = 0) Then
        ierror = 3
        Exit Sub
    End If

' Set the increment.
'
    dx = -fx / m
'
' Remember current data for next step.
'
    x2 = x1
    fx2 = fx1

    x1 = x
    fx1 = fx
'
' Update the iterate and function values.
'
    x = x + dx
    fx = Funct(x, 0)

Loop
End Sub

'*****
' STEFFENSON implements Steffenson's method.
'
' Parameters:
'
'   Input/output, real X.
'   On input, the point that starts the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR.
'
' Reference:
'
'   Joseph Traub,
'   Iterative Methods for the Solution of Equations,
'   Prentice Hall, 1964, page 178.
'*****

Sub steffenson(x, abserr, ierror, k, kmax)
'
Dim fx, gx, dx, y
'
' Initialization.
'
ierror = 0
k = 0
fx = Funct(x, 0)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
    If (Abs(fx) <= abserr) Then
        Exit Sub
    End If

    k = k + 1

```

```

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If
    y = x + fx
    gx = (Funct(y, 0) - fx) / fx

    If (gx = 0) Then
        ierror = 3
        Exit Sub
    End If
,
' Set the increment.
,
    dx = -fx / gx
,
' Update the iterate and function values.
,
    x = x + dx
    fx = Funct(x, 0)

Loop
End Sub

*****
' NEWTON implements Newton's method.
,
' Parameters:
,
'   Input/output, real X.
'   On input, an estimate for the root of the equation.
'   On output, if IERROR = 0, X is an approximate root for which
'   abs ( F(X) ) <= ABSERR.
*****

Sub newton(x, abserr, ierror, k, kmax)
,
Dim fx, dx, dfx, tol
,
' Initialization.
,
    ierror = 0
    k = 0
    fx = Funct(x, 0)
    tol = 5 * abserr
,
' Iteration loop:
,
Do
,
' If the error tolerance is satisfied, then exit.
,
    If (Abs(fx) <= tol) Then
        Exit Sub
    End If

    k = k + 1

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If

    dfx = Funct(x, 1)

    If (dfx = 0) Then
        ierror = 3
        Exit Sub
    End If
,
' Set the increment.
,
    dx = -fx / dfx
,
' Update the iterate and function values.

```

```

'
  x = x + dx
  fx = Funct(x, 0)

Loop
End Sub

'*****
'
' HALLEY_FD implements Halley's method, with finite differences.
'
' Parameters:
'
'   Input/output, real X, X1, X2.
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'   previous estimates.
'*****

Sub halley_fd(x, x1, x2, abserr, ierror, k, kmax)
Dim dx, fx, fx1, fx2, d1, d2, z
'
' Initialization.
'
  ierror = 0
  k = 0
  x = (x1 + x2) / 2
  fx = Funct(x, 0)
  fx1 = Funct(x1, 0)
  fx2 = Funct(x2, 0)

  d1 = (fx1 - fx2) / (x1 - x2)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
  If (Abs(fx) <= abserr) Then
    Exit Sub
  End If

  k = k + 1

  If (k > kmax) Then
    ierror = 2
    Exit Sub
  End If

  d2 = d1

  If (x = x1) Then
    ierror = 3
    Exit Sub
  End If

  d1 = (fx - fx1) / (x - x1)

  If (x = x2) Then
    ierror = 3
    Exit Sub
  End If

  d2 = (d1 - d2) / (x - x2)

  If (d1 = 0) Then
    ierror = 3
    Exit Sub
  End If

  z = d1 - fx1 * d2 / d1

  If (z = 0) Then

```

```

        ierror = 3
        Exit Sub
    End If
'
' Set the increment.
'
    dx = -fx / z
'
' Remember current data for next step.
'
    x2 = x1
    fx2 = fx1
    x1 = x
    fx1 = fx
'
' Update the iterate and function values.
'
    x = x + dx
    fx = Funct(x, 0)

Loop
End Sub

*****
' HALLEY1 implements Halley's method.
'
' Parameters:
'
'   Input/output, real X.
'   On input, an estimate for the root of the equation.
'   On output, if IERROR = 0, X is an approximate root for which
'   abs ( F(X) ) <= ABSERR.
'
*****

Sub halley1(x, abserr, ierror, k, kmax)
'
Dim fx, dfx, d2fx, dx, u, tol
'
' Initialization.
'
ierror = 0
k = 0
fx = Funct(x, 0)
dfx = Funct(x, 1)
d2fx = Funct(x, 2)
tol = 5 * abserr
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
    If (Abs(fx) <= tol) Then
        Exit Sub
    End If

    k = k + 1

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If

    If (dfx = 0) Then
        ierror = 3
        Exit Sub
    End If

    u = d2fx * fx / dfx ^ 2

    If (2 - u = 0) Then
        ierror = 4
        Exit Sub
    End If

```

```

'
' Set the increment.
'
'   dx = -(fx / dfx) / (1 - 0.5 * u)
'
' Update the iterate and function values.
'
'   x = x + dx
'   fx = Funct(x, 0)
'   dfx = Funct(x, 1)
'   d2fx = Funct(x, 2)
'
' Loop
'
End Sub

'*****
'
' SECANT_BACK implements the safe secant method with back-step
'
' Parameters:
'
'   Input/output, real X, a, b.
'   where F(a) * F(b) < 0
'
' Author: Leonardo Volpi
'
'*****

Sub secant_back(x, a, b, abserr, ierror, k, kmax)
'
Dim fx0, fx1, fx2, dx, x0, x1, x2, ferr
'
' Initialization.
'
'   ierror = 0
'   k = 0
'   x1 = a
'   x2 = b
'
'   fx1 = Funct(x1, 0)
'   fx2 = Funct(x2, 0)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
'   ferr = min(Abs(fx1), Abs(fx2))
'   If (ferr <= abserr) Then
'     Exit Sub
'   End If
'
'   k = k + 1
'
'   If (k > kmax) Then
'     ierror = 2
'     Exit Sub
'   End If
'
'   If ((fx1 - fx2) = 0#) Then
'     ierror = 3
'     Exit Sub
'   End If
'
' Set the increment.
'
'   dx = -fx2 * (x2 - x1) / (fx2 - fx1)
'   x = x2 + dx
'
'   If a < x And x < b Then
'     'accept the point x
'     x0 = x1: x1 = x2: x2 = x
'     fx0 = fx1: fx1 = fx2
'     fx2 = Funct(x2, 0)
'   Else

```

```

        'reject the point x
    If x0 = x1 Then
        ierror = 4
        Exit Sub 'nothing to do here
    End If
    x1 = x0
    fx1 = fx0
End If

Loop
x2 = x 'the best approximated root
End Sub

'*****
'
' parabola_inv implements the parabola inverse interpolation method.
'
' Parameters:
'
'   Input/output, real X, X1, X2.
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'   previous estimates.
'
' Author: Leonardo Volpi
'
'*****

Sub parabola_inv(x, x1, x2, abserr, ierror, k, kmax)
'
Dim y, y1, y2, dx, z
'
' Initialization.
'
ierror = 0
k = 0
x = (x1 + x2) / 2
y = Funct(x, 0)
y1 = Funct(x1, 0)
y2 = Funct(x2, 0)
'
' Iteration loop:
'
Do
'
' If the error tolerance is satisfied, then exit.
'
    If (Abs(y) <= abserr) Then
        Exit Sub
    End If

    k = k + 1

    If (k > kmax) Then
        ierror = 2
        Exit Sub
    End If

    If Abs(y - y1) <= abserr Then
        ierror = 3
        Exit Sub
    End If
'
' Set the increment.
z = (y1 * (x2 * y - x * y2) / (y2 - y) - y * (x1 * y2 - x2 * y1) / (y1 - y2)) / (y - y1)
'
' Remember current data for next step.
'
    x1 = x2
    y1 = y2

    x2 = x
    y2 = y
'
' Update the iterate and function values.

```

```

'
'   x = z
'   y = Funct(x, 0)
'
' Loop
'
' End Sub
'
' *****
'
' FRACTION implements the linear fraction interpolation method.
'
' Parameters:
'
'   Input/output, real X, X1, X2.
'   On input, three distinct points that start the method.
'   On output, X is an approximation to a root of the equation
'   which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'   previous estimates.
'
' Author: Leonardo Volpi
'
' *****
'
' Sub fraction(x, x1, x2, abserr, ierror, k, kmax)
'
' Dim y, y1, y2, dx, d
'
' Initialization.
'
'   ierror = 0
'   k = 0
'   x = (x1 + x2) / 2
'   y = Funct(x, 0)
'   y1 = Funct(x1, 0)
'   y2 = Funct(x2, 0)
'
' Iteration loop:
'
' Do
'
'   If the error tolerance is satisfied, then exit.
'
'   If (Abs(y) <= abserr) Then
'     Exit Sub
'   End If
'
'   k = k + 1
'
'   If (k > kmax) Then
'     ierror = 2
'     Exit Sub
'   End If
'
'   d = (y1 - y) / (x1 - x) * y2 - (y2 - y) / (x2 - x) * y1
'
'   If Abs(d) <= abserr Then
'     ierror = 3
'     Exit Sub
'   End If
'
' Set the increment.
'
'   dx = (y1 - y2) * y / d
'
' Remember current data for next step.
'
'   x1 = x2
'   y1 = y2
'
'   x2 = x
'   y2 = y
'
' Update the iterate and function values.
'
'   x = x + dx
'   y = Funct(x, 0)

```

```
Loop  
End Sub  
'  
' End of Zerofinder Miscellanea in VB  
'  
'*****
```




© Dic. 2005, by Foxes Team
ITALY
leovlp@libero.it

1. Edition