DUE ALGORITMI IN AUTOLISP PER LA TRILATERAZIONE PIANA IN AUTOCAD

Ottimizzazione numerica della soluzione e Visualizzazione volumetrica dell'errore

Giovanni Anzani − 2025
Università degli Studi di Firenze - Dipartimento di Architettura DIDA
Via Della Mattonaia 8 - 50121 Firenze (FI)
Mob.: +39/339/7056663
giovanni.anzani@unifi.it

ABSTRACT

Questo documento tecnico fornisce la documentazione completa di due programmi AutoLISP per l'ambiente AutoCAD che implementano approcci complementari alla soluzione del problema della trilaterazione nel piano bidimensionale.

- Listato 1 (68 funzioni, 410 righe di codice) implementa un sistema comparativo che esegue in serie quattro algoritmi di ottimizzazione numerica (Discesa del Gradiente, Newton-Raphson, Levenberg-Marquardt, Griglia Adattiva) utilizzando algebra lineare completa (gradiente, Hessiana, risoluzione di sistemi 3×3) per localizzare il punto ottimale a partire da tre circonferenze di riferimento. Il programma produce un report dettagliato—sia numerico che grafico—che permette di confrontare le prestazioni e l'accuratezza di ciascun metodo e risponde alla domanda: "quale algoritmo funziona meglio per il mio problema specifico?" consentendo una valutazione empirica dei trade-off tra velocità, robustezza e precisione.
- Listato 2 (43 funzioni, 260 righe di codice) realizza invece un sistema di visualizzazione volumetrica: la funzione di errore, calcolata rispetto alle tre circonferenze, viene trasformata in un modello tridimensionale. Grazie alla metafora orografica, permette così una lettura visiva e quantitativa del "paesaggio di errore". L'utente può selezionare una tra quattro metriche statistiche (Media Semplice, EQMR, Scarto Quadratico Medio, Indice Composto Errore-Scarto) da utilizzare come funzione obiettivo, rappresentate attraverso una pipeline in cinque fasi (griglia, spline, superficie, solido, sezionamento) che produce visualizzazioni tridimensionali differenti a seconda della scelta effettuata. Questo strumento risponde soprattutto alla domanda: "come si comporta la funzione di errore nello spazio candidato?" rendendo tangibile la metafora orografica e favorendo un'analisi visuale delle soluzioni.

Il presente documento tecnico costituisce il complemento implementativo dell'articolo che verrà pubblicato nella rubrica "Codici Grafici" della rivista Tribelon (Vol. 2 | N. 4 | 2025) dal titolo: Algoritmi AI per la trilaterazione: come ottimizzare e visualizzare l'errore in 3D. L'articolo introduce il problema della trilaterazione, descrive dettagliatamente le proprietà matematiche sia delle funzioni obiettivo sia di quelle di ottimizzazione, analizza la morfologia e l'orografia del paesaggio di errore con esempi numerici e collega la metrica ICES al problema di Apollonio e alla teoria delle bisettrici curvilinee. Se la pubblicazione fornisce la giustificazione teorica e l'interpretazione geometrica, il presente documento ne trasforma i concetti in implementazione operativa in linguaggio AutoLISP pronta all'uso e completamente documentata.

Struttura del documento

La documentazione è suddivisa in sei sezioni, dalla panoramica algoritmica generale sino alla consultazione tecnica di dettaglio:

- 2 **Implementazioni Algoritmiche** che descrivono l'architettura, i principi operativi e il flusso esecutivo dei due listati, offrendo una visione chiara delle scelte progettuali;
- 1 **Categorizzazione Funzionale** che classifica le 97 funzioni uniche in 11 categorie funzionali, con indicatori che ne documentano la presenza nei listati;
- 1 **Glossario Funzioni** che fornisce un glossario tecnico dettagliato e ordinato alfabeticamente di tutte le funzioni, chiarendo parametri, scopo e valore restituito;
- 2 **Codici dei Listati** che contengono infine il codice commentato dei due listati, pronto per essere utilizzato in AutoCAD.

Come utilizzare questa documentazione

La documentazione è pensata per sviluppatori AutoLISP, ingegneri CAD, ricercatori in geometria computazionale e utenti avanzati di AutoCAD interessati a tecniche di ottimizzazione geometrica e visualizzazione volumetrica. Si presuppone familiarità con la programmazione in AutoLISP, con l'ambiente AutoCAD e con i fondamenti di ottimizzazione numerica e geometria analitica.

- Per comprendere l'approccio generale: leggere le Implementazioni Algoritmiche che descrivono architettura, flusso operativo e logica dei due listati senza entrare nel dettaglio funzionale.
- Per navigare la struttura funzionale: consultare la Categorizzazione Funzionale che classifica le 97 funzioni in 11 categorie, evidenziando mediante indicatori [1] [2] [1+2] la presenza nei listati e identificando la base comune condivisa.
- Per consultazione tecnica di dettaglio: utilizzare il Glossario Funzioni come riferimento alfabetico completo di ogni funzione con firma, parametri, scopo operativo e valore restituito.
- Per utilizzo pratico: copiare il Codice dei listati 1 e 2 direttamente in AutoCAD e invocare i comandi TRILATCOMPARE e ANALISI dopo aver disegnato tre circonferenze.
- Per approfondimento teorico: consultare l'articolo autonomo correlato che fornisce fondamenti matematici, dimostrazioni formali e interpretazione geometrica dei risultati prodotti dai programmi.

Nota implementativa

I due listati sono stati sviluppati in AutoLISP per garantire massima integrazione con AutoCAD e per sfruttare le primitive geometriche native del CAD. L'approccio funzionale adottato (funzioni pure con input/output espliciti, assenza di stato globale condiviso) garantisce robustezza, manutenibilità e riutilizzabilità del codice. La struttura modulare con tre livelli di astrazione (utilità generale, operazioni specializzate, coordinamento di alto livello) permette di comprendere e modificare il codice progressivamente dal basso verso l'alto.

IMPLEMENTAZIONE ALGORITMICA 1

Il primo listato implementa un sistema comparativo di trilaterazione che risolve il problema della localizzazione ottimale di un punto nello spazio bidimensionale a partire da tre circonferenze di riferimento. Il software automatizza l'intero processo: dalla selezione interattiva delle tre circonferenze alla determinazione del punto di partenza (baricentro dei centri), dall'esecuzione parallela di quattro algoritmi di ottimizzazione distinti alla visualizzazione grafica dei risultati su layer dedicati, fino alla produzione di un report testuale comparativo completo.

Architettura del codice

Il listato è strutturato in 68 funzioni organizzate gerarchicamente su tre livelli di astrazione, per un totale di circa 410 righe di codice AutoLISP. Il livello inferiore comprende funzioni di utilità generale per geometria, algebra lineare e gestione dati DXF; il livello intermedio implementa i quattro motori di ottimizzazione con le rispettive funzioni matematiche di supporto (gradiente, Hessiana, risoluzione sistemi lineari); il livello superiore coordina il flusso operativo completo attraverso il comando principale C:TRILATCOMPARE e le funzioni di visualizzazione e analisi dei risultati.

I quattro algoritmi di ottimizzazione

Come descritto nell'articolo, il software implementa quattro strategie algoritmiche distinte per la minimizzazione della funzione obiettivo SSE (Somma dei Quadrati degli Errori), ciascuna con caratteristiche specifiche di convergenza, efficienza e robustezza. Gli algoritmi operano simultaneamente sullo stesso punto di partenza (baricentro dei centri delle tre circonferenze) e vengono confrontati in termini di accuratezza della soluzione finale, numero di iterazioni richieste, tempo di esecuzione e stato di convergenza.

- Discesa del Gradiente (GD) – Implementata nella funzione optimize-GD, utilizza derivate di primo ordine per muoversi iterativamente nella direzione di massima pendenza negativa con passo controllato da learning rate fisso (0.01).
- Newton-Raphson (NR) – Implementata nella funzione optimize-NR, sfrutta derivate di primo e secondo ordine (gradiente e Hessiana) per convergenza rapida mediante approssimazione quadratica locale, con gestione della singolarità della Hessiana.
- Levenberg-Marquardt (LM) – Implementata nella funzione optimize-LM, metodo ibrido che interpola dinamicamente tra GD e NR mediante parametro di smorzamento λ adattivo, bilanciando robustezza ed efficienza.
- Griglia Adattiva (GA) – Implementata nella funzione optimize-GA, strategia di ricerca esaustiva iterativa che raffina progressivamente una griglia di campionamento centrata sul minimo corrente, senza utilizzare derivate.

Funzione obiettivo e supporto matematico

La funzione obiettivo objective-function calcola la somma dei quadrati degli errori (SSE) tra le distanze del punto candidato dai centri delle circonferenze e i rispettivi raggi, misurando l'incoerenza geometrica complessiva. Le funzioni di supporto matematico implementano: il calcolo del gradiente mediante differenziazione numerica (gradient-function), la costruzione della matrice Hessiana 3×3 con derivate parziali seconde (hessian-function), la risoluzione di sistemi lineari 3×3 mediante regola di Cramer (solve-system-3x3, determinant-3x3), e operazioni di algebra lineare vettoriale e matriciale (somma, differenza, scalatura, norma).

Sistema di organizzazione grafica e output

I risultati delle quattro ottimizzazioni vengono visualizzati su layer dedicati con colorazione distintiva: TRILAT-GD (rosso, colore 1), TRILAT-LM (ciano, colore 4), TRILAT-NR (magenta, colore 5), TRILAT-GA (giallo, colore 6). Ogni punto calcolato viene rappresentato mediante entità POINT con stile e dimensione controllati dalle variabili di sistema pdmode (34, croce nel cerchio) e pdsize (-2.5, dimensione percentuale rispetto allo schermo). L'output testuale, generato dalle funzioni princ-advanced-results e correlate, presenta una tabella comparativa strutturata in undici sezioni che riportano: punti di partenza e finali, errori iniziali e finali, residui per ciascuna circonferenza, numero di iterazioni, stato di convergenza, durata temporale, identificazione del metodo migliore.

Flusso operativo del comando principale

Il comando C:TRILATCOMPARE orchestra l'intera procedura in sei fasi sequenziali: (1) selezione interattiva delle tre circonferenze mediante funzione select-three-circles con validazione del tipo di entità e gestione dell'annullamento; (2) estrazione dei parametri geometrici (centri e raggi) mediante Get_Data_LisCir; (3) calcolo del punto di partenza come baricentro dei centri mediante Media_lis_Pu; (4) esecuzione parallela delle quattro ottimizzazioni con creazione automatica dei layer e disegno dei punti risultanti; (5) costruzione delle strutture dati dei risultati mediante build-optimization-results; (6) visualizzazione del report comparativo completo e misurazione della durata totale di esecuzione mediante contatempo.

Utilità e funzioni di supporto generale

Il listato include un nucleo consistente di funzioni di utilità riutilizzabili per: gestione punti 2D/3D con predicati di validazione (P_Pu, P_Pu_2D, P_Pu_3D) e conversione (Pu_3D, Pu_2D); operazioni su liste di punti (baricentro, bounding box, coordinate minime/massime); costruzione codici DXF per entità AutoCAD (cons_0, cons_8, cons_10, cons_100); gestione layer (verifica esistenza, creazione con colore); formattazione output numerico con padding e precisione controllata (rtos-pad, itoa-pad); algebra vettoriale (somma, differenza, scala, norma) e matriciale (identità, scala, somma); calcolo distanze ed errori geometrici; misurazione temporale per analisi prestazionale.

IMPLEMENTAZIONE ALGORITMICA 2

Il secondo listato implementa un sistema di visualizzazione volumetrica che trasforma la funzione di errore di un problema di trilaterazione in un modello tridimensionale interpretabile mediante metafora orografica. Il software automatizza l'intero processo: dalla selezione interattiva delle tre circonferenze di riferimento alla scelta della funzione obiettivo statistica (MS, EQMR, SQM, ICES), dalla definizione della densità della griglia di campionamento alla costruzione della pipeline volumetrica completa (griglia di punti, rete di spline, superficie loftata, solido 3D, curve di livello per sezioni orizzontali), fino alla visualizzazione finale organizzata su layer dedicati con colorazione distintiva.

Architettura del codice

Il listato è strutturato in 43 funzioni organizzate gerarchicamente su tre livelli di astrazione, per un totale di circa 260 righe di codice AutoLISP. Il livello inferiore comprende circa 14 funzioni di utilità generale condivise con il Listato 1 (geometria, predicati, gestione layer, codici DXF); il livello intermedio implementa le quattro funzioni obiettivo statistiche, la generazione della griglia di errore, e le funzioni di disegno specializzate per entità geometriche complesse (spline, superfici, solidi); il livello superiore coordina il flusso operativo attraverso il comando principale C:ANALISI e la funzione di orchestrazione Sol_Analysis che gestisce la pipeline volumetrica completa.

Le quattro funzioni obiettivo statistiche

Come descritto nell'articolo, il software implementa quattro metriche statistiche alternative per la mappatura del paesaggio di errore, ciascuna con una specifica interpretazione geometrica e statistica. L'utente seleziona interattivamente quale funzione utilizzare, determinando così la morfologia della superficie tridimensionale risultante.

- Media Semplice (MS) – Implementata nella funzione MS, calcola la media aritmetica delle distanze dai tre perimetri. Funzione lineare che fornisce una misura diretta della distanza media senza enfatizzare gli outlier.
- Radice quadrata dell'Errore Quadratico Medio (EQMR) – Implementata nella funzione EQMR, corrisponde alla metrica Root Mean Square Error (RMSE) che penalizza fortemente gli scarti maggiori mediante elevazione al quadrato. Genera paesaggi con morfologia relativamente semplice (vulcano, catena montuosa, fondovalle) a seconda della configurazione geometrica delle circonferenze.
- Scarto Quadratico Medio (SQM) – Implementata nella funzione SQM, equivalente alla deviazione standard, misura la dispersione delle tre distanze rispetto alla loro media. Ignora la magnitudo dell'errore e focalizza l'attenzione sull'omogeneità delle distanze, identificando zone di equilibrio geometrico.
- Indice Composto Errore-Scarto (ICES) – Implementata nella funzione ICES, metrica ibrida originale definita come media geometrica di EQMR e SQM. Bilancia magnitudo e coerenza dell'errore, generando paesaggi articolati con molteplici vette di altezza comparabile che evidenziano regioni di buon compromesso geometrico. La struttura intrinseca della funzione è collegata al problema di Apollonio (CCC) e alla teoria delle bisettrici curvilinee.

La pipeline volumetrica in cinque fasi

La costruzione del modello tridimensionale procede attraverso una sequenza di trasformazioni geometriche coordinate dalla funzione Sol_Analysis, che implementa il flusso completo dalla griglia bidimensionale al solido tridimensionale sezionato.

- Fase 1: Generazione griglia di errore – La funzione Build_Grid calcola una griglia regolare di punti nel rettangolo di contenimento delle tre circonferenze, con densità selezionabile dall'utente (da 16×16 a 256×256 punti, corrispondenti a potenze di 2 da n=4 a n=8). Per ogni punto della griglia viene valutata la funzione obiettivo scelta mediante Test_Pu_LisCir, che calcola le distanze dai perimetri delle circonferenze e applica la metrica statistica. Il valore risultante viene negato e assegnato come coordinata z, trasformando i minimi della funzione di errore in massimi della superficie (vette del paesaggio).
- Fase 2: Costruzione rete di spline – La funzione Draw_Spl_From_Grid genera due famiglie ortogonali di curve spline (una per ogni riga e una per ogni colonna della griglia) mediante interpolazione cubica (grado 3) dei punti tridimensionali calcolati. Queste spline costituiscono una rete di sezioni che definisce la forma della superficie da costruire.
- Fase 3: Generazione superficie loftata – La funzione Draw_SurNet_From_Spl utilizza il comando nativo AutoCAD SURFNETWORK per creare una superficie NURBS complessa interpolando le due famiglie di spline. La superficie risultante rappresenta fedelmente l'orografia del paesaggio di errore. Le spline intermedie vengono eliminate per pulizia grafica, mantenendo solo la superficie.
- Fase 4: Costruzione solido volumetrico – La funzione Draw_Sol_From_Surnet genera un parallelepipedo di base mediante Draw_Box_LisPu e lo "affetta" con la superficie loftata utilizzando il comando SLICE. Il solido risultante ha come "tetto" la mappatura volumetrica dell'errore, creando un vero e proprio modello topografico tridimensionale manipolabile.
- Fase 5: Generazione curve di livello – La funzione Draw_Sections_From_Solid seziona il solido con piani orizzontali equidistanti mediante il metodo SectionSolid degli oggetti ActiveX. Vengono generate due serie di sezioni: sezioni complessive con passo maggiore (ogni 10 unità dell'ordine di grandezza del range z) e sezioni apicali con passo minore (ogni unità) per dettagliare la zona critica vicino al minimo. Le curve di livello risultanti permettono la lettura quantitativa dell'andamento dell'errore, analogamente alle isoipse di una carta topografica.

Sistema di organizzazione grafica e denominazione dinamica

I layer di destinazione vengono generati dinamicamente in base alla funzione obiettivo selezionata, incorporando nel nome la sigla della metrica (MS, EQMR, SQM, ICES). La struttura standardizzata prevede quattro layer per ogni analisi: Tril_[TIPO]_Superf per la superficie loftata (colore 40, arancione), Tril_[TIPO]_Solido per il solido risultante (colore 1, rosso), Tril_[TIPO]_SezCom per le sezioni complessive (colore 9, grigio chiaro), Tril_[TIPO]_SezApi per le sezioni apicali di dettaglio (colore 6, magenta). Questo sistema permette di eseguire analisi comparative con diverse funzioni obiettivo mantenendo organizzati i risultati su layer separati facilmente identificabili.

Gestione della densità e ottimizzazione computazionale

La densità della griglia influenza direttamente sia la qualità del modello tridimensionale sia il carico computazionale. L'interfaccia utente presenta una tabella informativa che mostra per ogni valore di n (da 4 a 8) la densità risultante ($2^n \times 2^n$) e il numero totale di punti da calcolare (2^{2n}). Un limitatore di sicurezza nella funzione Sol_Analysis vincola la densità massima a n=9 (512×512 punti) per prevenire blocchi del sistema durante la generazione del solido. Le funzioni OrdineDimensione e OrdineEQuantita analizzano automaticamente il range verticale della superficie per determinare il numero ottimale di sezioni da generare, garantendo curve di livello leggibili senza sovraccarico grafico.

Flusso operativo del comando principale

Il comando C:ANALISI orchestra l'intera procedura in sei fasi sequenziali marcate graficamente mediante separatori numerati generati da #----#: (1) selezione interattiva del tipo di analisi mediante Get_Tipo_Analisi con opzioni MS/EQMR/SQM/ICES e default EQMR; (2) selezione della densità della griglia mediante Get_Densita_Griglia con presentazione tabella informativa e scelta vincolata tra n=4 e n=8; (3) inizializzazione layer con colorazione e stampa riepilogo layer utilizzati; (4) selezione interattiva delle tre circonferenze di riferimento mediante Get_LisCir; (5) esecuzione completa della pipeline volumetrica attraverso Sol_Analysis con generazione superficie, solido e sezioni; (6) visualizzazione durata totale di esecuzione mediante contatempo e ripristino variabili di sistema (osmode, 3dosmode).

Funzioni di utilità specifiche e supporto geometrico

Il listato include funzioni specializzate per la manipolazione di entità geometriche complesse: disegno di spline NURBS con controllo di grado e chiusura (Draw_Spline), calcolo bounding box di entità mediante metodi ActiveX (Get_BouBox_Ent, Get_BouBox_LisEnt). Le funzioni di analisi numerica LisDec_Num, OrdineDimensione e OrdineEQuantita implementano logica per determinare automaticamente l'ordine di grandezza dei valori e ottimizzare la spaziatura delle sezioni orizzontali.

CATEGORIZZAZIONE FUNZIONALE

Le 97 funzioni totali implementate complessivamente nei due listati possono essere classificate in dieci categorie funzionali secondo il loro ruolo algoritmico e computazionale. Questa classificazione unificata, evidenzia sia la base comune di funzioni di utilità condivise (circa 14 funzioni), sia le specializzazioni specifiche: 54 funzioni per l'ottimizzazione numerica (Listato 1) e 29 funzioni per la visualizzazione volumetrica (Listato 2).

LEGENDA INDICATORI DI UTILIZZO

- [1] — Funzione presente solo nel Listato 1
- [2] — Funzione presente solo nel Listato 2
- [1+2] — Funzione comune a entrambi i listati

Funzioni di validazione (Predicati)

Cinque funzioni con prefisso P_ che verificano la validità e la conformità dei dati geometrici. Questi predicati garantiscono la robustezza del codice mediante controlli preventivi sui parametri, restituendo T (vero) se la condizione è soddisfatta, nil (falso) altrimenti.

- **P_ent [1+2]** — Verifica se un valore è un'entità AutoCAD valida di tipo ENAME esistente nel database del disegno.
- **P_Layer [1+2]** — Verifica l'esistenza di un layer nella tabella dei simboli del disegno AutoCAD, restituendo la sua definizione completa se esiste.
- **P_Pu [1+2]** — Predicato fondamentale che verifica se un valore è un punto valido (lista di 2 o 3 elementi numerici).
- **P_Pu_2D [1+2]** — Verifica se un valore è un punto bidimensionale valido (lista di esattamente 2 elementi numerici).
- **P_Pu_3D** [1+2] — Verifica se un valore è un punto tridimensionale valido (lista di esattamente 3 elementi numerici).
- **P_tol** [1] — Verifica se due punti tridimensionali sono sufficientemente vicini rispetto a una tolleranza specificata, controllando sia le differenze coordinate per coordinata sia la distanza euclidea complessiva. Utilizzata dall'algoritmo GA per determinare la convergenza.

Funzioni di gestione layer

Due funzioni che implementano il sistema di organizzazione grafica per layer, fondamentale per la visualizzazione dei risultati in entrambi i listati.

- **Lay_Check** [1+2] — Verifica l'esistenza di un layer nel disegno e lo crea automaticamente con colore di default (7=bianco) se non esiste. Funzione di sicurezza utilizzata prima di disegnare entità su layer specifici.
- **Lay_Make_Col** [1+2] — Crea un nuovo layer con colore specificato se non esiste, oppure modifica il colore di un layer esistente. Utilizzata per inizializzare i layer di output con colorazione distintiva in entrambi i listati.

Funzioni di costruzione codici DXF

Cinque funzioni con prefisso cons_ che generano coppie DXF (codice-valore) per la costruzione programmatica di entità AutoCAD. Queste funzioni di basso livello forniscono i building block per assemblare le liste DXF complete passate a entmakex.

- **cons_0** [1+2] — Restituisce la coppia DXF (0 . "NOME_ENTITA") appropriata per il codice specificato (1-9): POLYLINE, VERTEX, SEQEND, LWPOLYLINE, LINE, CIRCLE, POINT, SPLINE, ELLIPSE. Definisce il tipo geometrico dell'entità.
- **cons_8** [1+2] — Restituisce la coppia DXF (8 . "NOME_LAYER") che assegna l'entità al layer specificato.
- **cons_10** [1] — Restituisce la coppia DXF (10 . (x y z)) che definisce il primo punto geometrico dell'entità. Garantisce tridimensionalità chiamando Pu_3D. Utilizzata nel Listato 1 per creare entità POINT.
- **cons_11** [2] — Restituisce la coppia DXF (11 . (x y z)) che definisce il secondo punto geometrico o punto direttore dell'entità. Garantisce tridimensionalità chiamando Pu_3D. Utilizzata nel Listato 2 per definire punti di controllo delle spline.
- **cons_100** [1+2] — Restituisce la coppia DXF (100 . "NOME_CLASSE") che definisce la classe di appartenenza dell'entità nella gerarchia ObjectARX: AcDbEntity, AcDb3dPolyline, AcDbVertex, AcDb3dPolylineVertex, AcDbPolyline, AcDbSpline, AcDbEllipse.

Funzioni di disegno e costruzione geometrica

Sette funzioni che creano entità geometriche nel disegno AutoCAD, dalle primitive semplici alle geometrie complesse necessarie per la pipeline volumetrica.

- **Draw_Point** [1] — Crea un'entità di tipo POINT alle coordinate specificate sul layer indicato. Utilizzata nel Listato 1 per visualizzare i quattro punti risultanti dalle ottimizzazioni GD, LM, NR, GA su layer distinti.
- **Draw_Spline** [2] — Crea una spline NURBS parametrica definita da lista di punti di controllo, con chiusura e grado specificati. Utilizzata nel Listato 2 per generare le curve interpolanti della griglia di errore nella Fase 2 della pipeline volumetrica.
- **Draw_Spl_From_Grid** [2] — Genera due famiglie ortogonali di spline (per righe e per colonne) dalla griglia tridimensionale di punti di errore. Crea la rete di curve che definisce la superficie nella pipeline volumetrica del Listato 2.
- **Draw_SurNet_From_Spl** [2] — Crea una superficie loftata NURBS dalla rete di spline mediante comando nativo AutoCAD SURFNETWORK. Fase 3 della pipeline

volumetrica. Elimina opzionalmente le spline intermedie per pulizia grafica.

- **Draw_Box_LisPu** [2] — Crea un solido parallelepipedo 3D definito da due punti opposti mediante comando nativo BOX. Utilizzata come geometria di base per l'operazione di slice nella Fase 4 della pipeline volumetrica.
- **Draw_Sol_From_Surnet** [2] — Costruisce il solido volumetrico creando un parallelepipedo di base e "affettandolo" con la superficie mediante comando SLICE. Fase 4 della pipeline. Elimina automaticamente la parte inferiore mantenendo solo il solido con "tetto" della superficie di errore, confrontando le quote dei centroidi.
- **Draw_Sections_From_Solid** [2] — Genera curve di livello sezionando il solido con piani orizzontali equidistanti mediante metodo ActiveX SectionSolid su oggetto VLA. Fase 5 della pipeline volumetrica. Crea serie di sezioni con passo specificato per visualizzazione dettagliata.

Funzioni di estrazione dati geometrici

Cinque funzioni che interrogano le entità AutoCAD estraendone i dati geometrici caratteristici necessari per i calcoli.

- **Get_DxfVal_Ent** [1+2] — Estrae il valore di un attributo DXF specifico da un'entità dato il codice numerico. Wrapper semplificato per accesso agli attributi DXF utilizzato pervasivamente in entrambi i listati.
- **Get_LisVal_LisAssoc** [1+2] — Estrae i valori corrispondenti a una lista di chiavi da una lista associativa. Funzione di utilità per estrarre multipli attributi DXF o elementi di strutture dati in una sola chiamata.
- **Get_Data_Cir** [1] — Estrae i dati geometrici di un cerchio (centro codice DXF 10 e raggio codice DXF 40). Utilizzata nel Listato 1 per estrarre dati dalle tre circonferenze selezionate dall'utente.
- **Get_Data_LisCir** [1] — Estrae i dati geometrici da una lista di cerchi applicando Get_Data_Cir a ciascuno e riorganizzando i risultati in due liste separate (centri e raggi). Utilizzata nel Listato 1 per organizzare i dati delle tre circonferenze di riferimento.
- **Get_BouBox_Ent** [2] — Calcola il bounding box (rettangolo di contenimento 3D) di un'entità mediante metodo ActiveX GetBoundingBox su oggetto VLA. Restituisce lista di due punti (minimo e massimo). Utilizzata nel Listato 2 per determinare la regione di ricerca.
- **Get_BouBox_LisEnt** [2] — Calcola il bounding box complessivo di una lista di entità applicando Get_BouBox_Ent a ciascuna e calcolando l'unione dei rettangoli. Utilizzata nel Listato 2 per definire il dominio della griglia di errore che racchiude tutte e tre le circonferenze.

Funzioni di calcolo matematico (Funzioni obiettivo ed errori)

Dieci funzioni che implementano le funzioni obiettivo e le metriche statistiche utilizzate per ottimizzazione (Listato 1) e visualizzazione (Listato 2).

- **objective-function** [1] — Calcola la somma dei quadrati degli errori (SSE) tra le distanze del punto candidato dai tre centri e i rispettivi raggi. Funzione obiettivo fondamentale che tutti e quattro gli algoritmi di ottimizzazione del Listato 1 mirano a minimizzare.
- **compute-error** [1] — Calcola l'errore quadratico medio (EQMR/RMSE) come radice quadrata della media dei quadrati degli errori. Utilizzata nel Listato 1 per valutare la qualità delle soluzioni finali e iniziali nelle tabelle comparative.
- **compute-residuals** [1] — Calcola i residui individuali per ciascuna delle tre circonferenze come differenza tra distanza dal centro e raggio. Utilizzata nel Listato 1 per l'analisi dettagliata dell'errore nelle tabelle di output, mostrando come l'errore si distribuisce tra le tre misurazioni.
- **MS** [2] — Calcola la Media Semplice di una lista di valori (media aritmetica). Prima delle quattro metriche statistiche disponibili nel Listato 2. Funzione obiettivo lineare che misura la distanza media dai perimetri senza enfatizzare outlier.
- **EQMR** [2] — Calcola la Radice quadrata dell'Errore Quadratico Medio (Root Mean Square Error). Seconda metrica statistica del Listato 2. Funzione obiettivo che penalizza fortemente gli scarti maggiori mediante elevazione al quadrato, genera paesaggi con morfologia relativamente semplice.
- **SQM** [2] — Calcola lo Scarto Quadratico Medio equivalente alla deviazione standard. Terza metrica statistica del Listato 2. Misura la dispersione delle distanze rispetto alla loro media ignorando la magnitudo dell'errore, evidenzia zone di equilibrio geometrico.
- **ICES** [2] — Calcola l'Indice Composto tra Errore e Scarto come media geometrica di EQMR e SQM. Quarta metrica del Listato 2. Metrica ibrida originale che bilancia magnitudo e coerenza dell'errore, genera paesaggi articolati con molteplici vette di altezza comparabile.
- **Test_Pu_LisCir** [2] — Valuta la funzione obiettivo scelta (MS/EQMR/SQM/ICES identificata da codice numerico 1-4) per un punto candidato rispetto alle tre circonferenze. Calcola le distanze dai perimetri mediante metodo ActiveX vlax-curve-getClosestPointTo e applica la metrica statistica selezionata. Funzione chiave per la generazione della griglia di errore nella Fase 1 della pipeline volumetrica.
- **Build_Grid** [2] — Genera la griglia tridimensionale di punti di errore campionando la funzione obiettivo scelta su una griglia regolare nel bounding box delle tre circonferenze. Fase 1 della pipeline volumetrica del Listato 2. Nega i valori di errore per trasformare minimi in massimi (metafora orografica: vette = soluzioni ottimali).
- **Sol_Analysis** [2] — Funzione di coordinamento di alto livello che orchestra l'intera pipeline volumetrica del Listato 2 in cinque fasi: generazione griglia, costruzione spline, creazione

superficie, generazione solido, sezioni orizzontali. Gestisce anche l'ottimizzazione del numero di sezioni mediante analisi dell'ordine di grandezza del range verticale con limitatore di sicurezza (densità massima n=9).

Funzioni di algebra lineare e manipolazione vettoriale

Dodici funzioni che implementano operazioni di algebra lineare vettoriale e matriciale, fornendo i building block matematici per gli algoritmi di ottimizzazione avanzati del Listato 1.

- **Somm_2Lis** [1] — Somma elemento per elemento di due liste (vettori). Operazione fondamentale utilizzata per aggiornare il punto candidato sommando il passo di ottimizzazione negli algoritmi GD, NR, LM.
- **Diff_2Lis** [1] — Differenza elemento per elemento di due liste (vettori). Utilizzata per calcolare vettori spostamento e differenze tra punti nel calcolo del gradiente e della Hessiana.
- **vector-scale** [1] — Moltiplica un vettore per uno scalare, restituendo un nuovo vettore con tutti gli elementi scalati. Utilizzata per controllare l'ampiezza dei passi di ottimizzazione (learning rate in GD, smorzamento in LM).
- **vector-length** [1] — Calcola la lunghezza (norma euclidea) di un vettore. Utilizzata per verificare la convergenza controllando se la norma del gradiente è sotto la soglia di tolleranza negli algoritmi GD, NR, LM.
- **matrix-identity** [1] — Genera una matrice identità quadrata di dimensione specificata. Utilizzata nell'algoritmo LM per costruire la matrice modificata H + λI che interpola tra comportamento GD e NR.
- **matrix-scale** [1] — Moltiplica una matrice per uno scalare, applicando vector-scale a ogni riga. Utilizzata in LM per scalare la matrice identità con il parametro di smorzamento λ.
- **matrix-sum** [1] — Somma due matrici elemento per elemento applicando Somm_2Lis a coppie di righe corrispondenti. Utilizzata in LM per costruire la matrice Hessiana modificata sommando H e λI.
- **gradient-function** [1] — Calcola il vettore gradiente (derivate parziali prime) della funzione obiettivo nel punto specificato. Utilizzata da GD, NR e LM per determinare la direzione di discesa ottimale. Gestisce la singolarità quando la distanza da un centro è quasi nulla (soglia 1e-10).
- **hessian-function** [1] — Calcola la matrice Hessiana 3×3 (derivate parziali seconde) della funzione obiettivo nel punto specificato. Utilizzata da NR e LM per approssimare localmente la funzione con una forma quadratica e determinare il passo ottimale. Gestisce la singolarità evitando divisioni per zero (soglia 1e-10).
- **determinant-3x3** [1] — Calcola il determinante di una matrice 3×3 mediante la regola di Sarrus. Utilizzata per verificare l'invertibilità della Hessiana in NR e come passo intermedio nella risoluzione di sistemi lineari mediante regola di Cramer.
- **solve-system-3x3** [1] — Risolve il sistema lineare Ax=b con matrice 3×3 mediante regola di Cramer calcolando quattro determinanti. Utilizzata in NR per calcolare il passo di Newton risolvendo H·step = -grad. Gestisce il caso di matrice singolare (determinante < 1e-10) restituendo un passo di fallback proporzionale al gradiente scalato.
- **range** [1+2] — Genera una lista di numeri interi consecutivi da a a b inclusi mediante iterazione e accumulo. Funzione di utilità per cicli e generazione indici, utilizzata pervasivamente in entrambi i listati per iterazioni su elementi di liste e matrici.

Funzioni di manipolazione punti e geometria

Undici funzioni che operano su punti singoli e liste di punti, fornendo utilità geometriche fondamentali per entrambi i listati.

- **Pu_3D** [1+2] — Garantisce la tridimensionalità di un punto: se il punto è 2D aggiunge coordinata z=0.0, se è già 3D lo restituisce invariato. Utilizzata pervasivamente per normalizzazione input e compatibilità tra funzioni in entrambi i listati.
- **Pu_2D** [1] — Converte un punto in formato 2D forzando z=0.0 mantenendo x e y. Utilizzata nel Listato 1 nelle funzioni di calcolo errore che operano esclusivamente sul piano xy.
- **Media_lis_Pu** [1] — Calcola il baricentro di una lista di punti mediando le coordinate x e y, utilizzando il valore assoluto per z. Utilizzata nel Listato 1 per calcolare il punto di partenza delle ottimizzazioni come baricentro dei centri delle tre circonferenze.
- **NCorMin_LisPu** [1+2] — Estrae la coordinata minima lungo l'asse n-esimo (0=x, 1=y, 2=z) da una lista di punti applicando min alla lista di coordinate estratte. Utilizzata per calcolare il bounding box in entrambi i listati.
- **NCorMax_LisPu** [1+2] — Estrae la coordinata massima lungo l'asse n-esimo da una lista di punti applicando max alla lista di coordinate estratte. Utilizzata per calcolare il bounding box in entrambi i listati.
- **PuMin_LisPu** [1+2] — Calcola il punto con le coordinate minime di una lista di punti applicando NCorMin_LisPu a ciascun asse (0, 1, 2). Restituisce il vertice minimo del bounding box in entrambi i listati.
- **PuMax_LisPu** [1+2] — Calcola il punto con le coordinate massime di una lista di punti applicando NCorMax_LisPu a ciascun asse. Restituisce il vertice massimo del bounding box in entrambi i listati.
- **LisPuBox_LisPu** [1+2] — Calcola il bounding box (rettangolo di contenimento 3D) di una lista di punti restituendo lista di due punti (minimo e massimo). Utilizzata per definire regioni di ricerca in entrambi i listati.
- **LisPuQuad_Cir** [1] — Genera i quattro punti cardinali (0°, 90°, 180°, 270°) sulla circonferenza di un cerchio mediante funzione polar applicata a quattro angoli. Utilizzata nel Listato 1 per campionare la geometria delle circonferenze di

riferimento e determinare il bounding box iniziale per l'algoritmo GA.

- **Rect_Ini** [1] — Calcola il rettangolo iniziale di ricerca per GA come bounding box dell'unione dei punti cardinali delle tre circonferenze. Applica LisPuQuad_Cir a ciascuna circonferenza e calcola LisPuBox_LisPu sui risultati concatenati. Definisce la regione candidata iniziale che racchiude tutte e tre le circonferenze.
- **Rect_Gen** [1] — Genera un nuovo rettangolo di ricerca centrato sul punto di minimo corrente con dimensioni ridotte secondo un fattore di raffinamento (multipl/divisions). Utilizzata da GA per implementare il meccanismo di "zoom" progressivo verso il minimo nelle iterazioni successive.

Funzioni di ottimizzazione (Algoritmi principali)

Cinque funzioni che implementano i motori di ottimizzazione del Listato 1. Ciascuna riceve le tre circonferenze, un punto di partenza (o solo le circonferenze per GA), un layer di destinazione, e restituisce una struttura dati completa con i risultati.

- **optimize-GD** [1] — Implementa l'algoritmo Discesa del Gradiente con learning rate fisso (0.01) e tolleranza sul gradiente (1e-6). Itera fino a 1000 volte o fino a convergenza verificata dalla norma del gradiente. Disegna il punto risultante sul layer specificato e restituisce struttura dati completa costruita da build-optimization-results.
- **optimize-NR** [1] — Implementa l'algoritmo Newton-Raphson con calcolo di gradiente e Hessiana ad ogni iterazione. Gestisce singolarità della Hessiana (determinante < 1e-10) riducendo il passo a fallback. Confronta errori per garantire miglioramento, altrimenti riduce il passo a metà. Tollera fino a 1000 iterazioni con soglia gradiente 1e-6.
- **optimize-LM** [1] — Implementa l'algoritmo Levenberg-Marquardt con parametro di smorzamento λ adattivo (iniziale 0.01, fattore moltiplicativo 10). Modifica dinamicamente λ: lo diminuisce se l'errore migliora (comportamento NR veloce), lo aumenta se peggiora (comportamento GD robusto). Tollera fino a 1000 iterazioni con soglia gradiente 1e-6.
- **optimize-GA** [1] — Implementa l'algoritmo Griglia Adattiva con ricerca esaustiva iterativa. Inizia con griglia 64×64 (interv=32, *2) sul bounding box delle circonferenze calcolato da Rect_Ini, poi raffina con griglie 32×32 progressivamente più piccole (apert=2) e centrate sul minimo corrente mediante Rect_Gen e calc_grid_err. Criterio di convergenza basato su vicinanza spaziale verificata da P_tol (tolleranza 1e-6) anziché su gradiente. Tollera fino a 40 cicli di raffinamento.
- **calc_grid_err** [1] — Valuta la funzione obiettivo su una griglia regolare di punti definita dal rettangolo (rect-min, rect-max) e dal numero di divisioni. Campiona la griglia chiamando compute-error per ogni punto e restituisce i due punti estremi: quello con errore minimo (candidato soluzione con z negativa) e quello con errore massimo. Utilizzata iterativamente da GA per identificare e raffinare la regione più promettente.

Funzioni di visualizzazione, input e output

Venticinque funzioni che gestiscono l'interfaccia utente, la selezione interattiva, la formattazione dell'output testuale e la presentazione dei risultati.

- **Get_LisCir** [2] — Gestisce la selezione interattiva guidata di un numero parametrico di circonferenze. Richiede ripetutamente la selezione verificando che siano entità di tipo CIRCLE mediante Get_DxfVal_Ent, consente annullamento in qualsiasi momento, fornisce feedback visivo con vl-princ-to-string dell'ENAME. Utilizzata nel Listato 2 per acquisire le tre circonferenze di riferimento.
- **select-three-circles** [1] — Versione specializzata fissa per tre circonferenze utilizzata nel Listato 1. Gestisce la selezione interattiva guidata delle tre circonferenze di riferimento con messaggi specifici e validazione tipo CIRCLE mediante Get_DxfVal_Ent.
- **contatempo** [1+2] — Misura la durata di esecuzione mediante variabile di sistema DATE. Alla prima chiamata (con nil) registra il tempo corrente e lo restituisce; alla seconda chiamata calcola la differenza in secondi (moltiplicata per 86400 per conversione da giorni), visualizza il risultato formattato in minuti e secondi e restituisce la durata numerica. Utilizzata in entrambi i listati per analisi prestazionale di singole ottimizzazioni e dell'intero comando.
- **#----#** [2] — Funzione di utilità grafica per output nella riga di comando. Stampa una riga di separazione orizzontale decorata con trattini, opzionalmente numerata per identificare le fasi. Utilizzata nel Listato 2 per separare visivamente le sei fasi di esecuzione del comando C:ANALISI.
- **Ent_Erase** [1+2] — Elimina dal database del disegno AutoCAD l'entità specificata mediante comando entdel. Verifica preventivamente che il parametro sia di tipo ENAME. Funzione di utilità per la pulizia di entità temporanee o indesiderate in entrambi i listati.
- **format-point** [1] — Formatta un punto come stringa leggibile nel formato "(x, y)" o "(x, y, z)" con precisione controllata e padding mediante rtos-pad. Utilizzata nel Listato 1 per visualizzare coordinate nelle tabelle di output comparative.
- **make-spaces** [1] — Genera una stringa di i spazi mediante iterazione repeat e concatenazione strcat. Utilizzata nel Listato 1 per padding e allineamento colonne nelle tabelle di output.
- **make-zeros** [1] — Genera una stringa di i zeri mediante iterazione repeat e concatenazione strcat. Utilizzata nel Listato 1 per completare la parte decimale dei numeri nelle formattazioni di rtos-pad.
- **itoa-pad** [1] — Converte un intero in stringa con padding di spazi a sinistra per raggiungere larghezza specificata. Utilizza make-spaces per calcolare il padding necessario. Utilizzata nel Listato 1 per allineare numeri di iterazioni nelle tabelle comparative.

- **rtos-pad** [1] — Converte un numero reale in stringa con modalità, precisione decimale e larghezza totale specificate, aggiungendo padding di spazi a sinistra e zeri a destra dove necessario. Funzione complessa di formattazione che gestisce anche il caso di numeri senza parte decimale. Utilizzata pervasivamente nel Listato 1 per allineare valori numerici nelle tabelle comparative.
- princ-results-row [1] — Stampa la prima sezione della tabella comparativa del Listato 1 mostrando per ogni metodo: nome, punto di partenza, errore iniziale, punto finale, errore finale. Estrae i dati dalle strutture associative mediante Get_LisVal_LisAssoc e formatta i dati in colonne allineate con format-point e rtos-pad.
- **princ-best-method** [1] — Identifica e annuncia il metodo con l'errore finale minore nel Listato 1 mediante iterazione su tutti i risultati e confronto numerico. Fornisce indicazione chiara della soluzione migliore tra le quattro calcolate.
- **princ-methods-with-convergence** [1] — Stampa la sezione di analisi della convergenza del Listato 1 mostrando per ogni metodo: stato di convergenza (SI/NO), durata calcolo formattata in minuti e secondi, numero di iterazioni con padding. Estrae i dati dalle strutture associative e formatta con itoa-pad e rtos-pad.
- **princ-radius** [1] — Stampa i raggi delle tre circonferenze di riferimento e il raggio medio nel Listato 1. Estrae la lista raggi dalla prima struttura risultati e calcola media aritmetica. Fornisce contesto geometrico per interpretare gli errori.
- **princ-residuals-by-method** [1] — Stampa la sezione di analisi dettagliata degli errori del Listato 1 mostrando per ogni metodo: i tre residui individuali (uno per circonferenza) e la media dei valori assoluti. Estrae i dati dalle strutture associative e formatta con rtos-pad per allineamento colonne.
- **-----** [1] — Genera una riga di separazione con trattini ripetuti (num × 5 caratteri) mediante iterazione repeat e concatenazione strcat. Utilizzata nel Listato 1 per delimitare le sezioni della tabella di output.
- **=====** [1] — Genera una riga di separazione enfatica con segni di uguale ripetuti (num × 5 caratteri) mediante iterazione repeat e concatenazione strcat. Utilizzata nel Listato 1 per intestazioni principali della tabella di output.
- **princ-advanced-results** [1] — Funzione di coordinamento di alto livello che orchestra la stampa dell'intero report comparativo del Listato 1 in undici sezioni strutturate. Invoca nell'ordine: intestazione con =====, riga risultati con princ-results-row, convergenza con princ-methods-with-convergence, metodo migliore con princ-best-method, raggi con princ-radius, residui con princ-residuals-by-method. Utilizza separatori grafici ----- e ===== per delimitare le sezioni.
- **build-optimization-results** [1] — Costruisce e restituisce una lista associativa (struttura dati) contenente tutti i risultati di un'ottimizzazione nel Listato 1: lista raggi, punto iniziale, punto finale, numero iterazioni, errore iniziale (calcolato da compute-error), errore finale (calcolato da compute-error), residui (calcolati da compute-residuals), convergenza (booleano), storia errori (lista invertita), parametro lambda finale (nil per GD/NR/GA), durata, nome metodo. Questa struttura standardizzata con chiavi simboliche permette l'elaborazione uniforme dei risultati dei quattro algoritmi mediante Get_LisVal_LisAssoc.
- **Get_Tipo_Analisi** [2] — Gestisce la selezione interattiva del tipo di funzione obiettivo mediante keyword vincolate con initget e getkword. Opzioni: MS/EQMR/SQM/ICES con default EQMR. Restituisce codice numerico 1-4 corrispondente alla scelta. Utilizzata nella Fase 1 del comando C:ANALISI per determinare quale metrica statistica utilizzare nella mappatura.
- **Get_Densita_Griglia** [2] — Gestisce la selezione interattiva della densità della griglia presentando una tabella informativa ASCII-art che mostra per ogni valore di n (4-8) la densità risultante ($2^n \times 2^n$) e il numero totale di punti (2^{2n}). Valida l'input vincolando a valori 4-8 con default 6 (64×64). Utilizzata nella Fase 2 del comando C:ANALISI.
- **LisDec_Num** [2] — Genera la lista delle potenze di 10 dalla unità fino all'ordine di grandezza superiore al numero specificato mediante iterazione moltiplicativa (es. per 3456 → [1, 10, 100, 1000, 10000]). Utilizzata nel Listato 2 per analisi dell'ordine di grandezza.
- **OrdineDimensione** [2] — Determina l'ordine di grandezza (potenza di 10) di un numero mediante iterazione sulla lista generata da LisDec_Num e selezione della potenza massima non superiore al numero. Utilizzata nel Listato 2 per calcolare automaticamente il passo delle sezioni orizzontali.
- **OrdineEQuantita** [2] — Calcola sia l'ordine di grandezza (mediante OrdineDimensione) sia il numero di "unità" di quell'ordine contenute nel numero assoluto (mediante divisione e fix). Restituisce lista di due elementi: ordine e quantità. Utilizzata nel Listato 2 per ottimizzare il numero di sezioni da generare nella Fase 5 della pipeline volumetrica.

Comandi principali

Due funzioni comando che coordinano l'intero flusso operativo dei rispettivi listati, orchestrando chiamate sequenziali a tutte le funzioni di supporto.

- **C:TRILATCOMPARE** [1] — Funzione comando principale del Listato 1 che implementa l'intera procedura comparativa di trilaterazione. Coordina l'esecuzione in fasi: messaggio introduttivo, selezione tre circonferenze mediante select-three-circles, inizializzazione quattro layer con colorazione mediante Lay_Make_Col (TRILAT-GD rosso 1, TRILAT-LM ciano 4, TRILAT-NR magenta 5, TRILAT-GA giallo 6), calcolo baricentro come punto di partenza mediante Media_lis_Pu sui centri estratti da Get_Data_LisCir, esecuzione parallela delle quattro ottimizzazioni (GD, LM, NR passano il baricentro; GA calcola autonomamente il punto iniziale), raccolta quattro strutture risultati, visualizzazione report comparativo completo

mediante princ-advanced-results. Gestisce la misurazione temporale complessiva mediante doppia chiamata a contatempo (inizio e fine). Ripristina CMDECHO a 0 e restituisce controllo con princ silenzioso.

- **C:ANALISI** [2] — Funzione comando principale del Listato 2 che implementa l'intera procedura di visualizzazione volumetrica del paesaggio di errore. Coordina l'esecuzione in sei fasi marcate graficamente da #----#: (1) selezione tipo analisi mediante Get_Tipo_Analisi, determinazione nome tipo (MS/EQMR/SQM/ICES) per layer dinamici, stampa conferma; (2) selezione densità griglia mediante Get_Densita_Griglia, stampa densità calcolata ($2^n \times 2^n$); (3) costruzione nomi layer dinamici mediante concatenazione (Tril_[TIPO]_Superf/Solido/SezCom/SezApi), inizializzazione layer con colorazione mediante Lay_Make_Col (Superf arancione 40, Solido rosso 1, SezCom grigio 9, SezApi magenta 6), stampa riepilogo layer; (4) selezione tre circonferenze mediante Get_LisCir; (5) esecuzione pipeline volumetrica completa mediante Sol_Analysis passando circonferenze, densità, tipo analisi, flag sezioni T, quattro layer; (6) visualizzazione durata mediante contatempo, ripristino variabili sistema (osmode, 3dosmode). Gestisce backup e ripristino variabili osmode e 3dosmode per garantire corretta esecuzione comandi nativi AutoCAD. Restituisce controllo con princ silenzioso.

GLOSSARIO FUNZIONI

Simboli

- **#----# (num)**

Parametri:

num – Numero intero opzionale per identificare la fase (può essere nil)

Scopo: Funzione di utilità grafica per output formattato nella riga di comando. Stampa una riga di separazione orizzontale decorativa composta da trattini, con numerazione opzionale alle estremità per identificare visivamente le fasi di esecuzione. La lunghezza della riga è fissa (circa 100 caratteri). Se num è fornito e minore di 10, aggiunge un trattino extra per compensazione visiva. Utilizzata nel comando C:ANALISI del Listato 2 per separare graficamente le sei fasi sequenziali dell'esecuzione (selezione tipo analisi, densità griglia, layer, circonferenze, pipeline, conclusione).

Valore restituito: Nessuno (funzione di side-effect che stampa su console).

- **----- (num)**

Parametri:

num – Numero intero che indica quante ripetizioni del pattern di base (5 trattini)

Scopo: Genera e stampa una riga di separazione composta da trattini ripetuti. Ogni unità num produce 5 caratteri "-", quindi num=24 genera 120 trattini totali. Utilizzata nel comando C:TRILATCOMPARE del Listato 1 per delimitare le sezioni intermedie della tabella comparativa di output (separa intestazioni, righe dati, analisi convergenza, metodo migliore, raggi, residui). Costruisce la stringa mediante iterazione con repeat e concatenazione con strcat, iniziando con "\n" per garantire separazione verticale.

Valore restituito: Stringa contenente "\n" seguito da (num × 5) caratteri "-".

- **===== (num)**

Parametri:

num – Numero intero che indica quante ripetizioni del pattern di base (5 segni di uguale)

Scopo: Genera e stampa una riga di separazione enfatica composta da segni di uguale ripetuti. Ogni unità num produce 5 caratteri "=", quindi num=24 genera 120 segni di uguale totali. Utilizzata nel comando C:TRILATCOMPARE del Listato 1 per delimitare le sezioni principali della tabella comparativa di output: intestazione iniziale del report e chiusura finale. La maggiore visibilità rispetto a "-----" serve a distinguere graficamente le sezioni di massimo livello gerarchico. Costruisce la stringa mediante iterazione con repeat e concatenazione con strcat, iniziando con "\n".

Valore restituito: Stringa contenente "\n" seguito da (num × 5) caratteri "=".

B

- **build-optimization-results (Pui Puf LisCen LisRad iter-num converged lerr lambda-param duration NameMethod)**

Parametri:

Pui – Punto tridimensionale iniziale dell'ottimizzazione (baricentro dei centri per GD/NR/LM, punto calcolato per GA)

Puf – Punto tridimensionale finale risultante dall'ottimizzazione

LisCen – Lista di tre punti (centri delle circonferenze di riferimento)

LisRad – Lista di tre numeri reali (raggi delle circonferenze di riferimento)

iter-num – Numero intero di iterazioni eseguite

converged – Booleano (T se convergenza raggiunta, nil altrimenti)

lerr – Lista degli errori intermedi accumulati durante le iterazioni (ordine inverso)

lambda-param – Parametro lambda finale (numero reale per LM, nil per GD/NR/GA)

duration – Durata dell'esecuzione in secondi (numero reale)

NameMethod – Stringa identificativa del metodo (es. "Discesa Gradiente " con padding)

Scopo: Costruisce e restituisce una lista associativa standardizzata contenente tutti i risultati di un'ottimizzazione nel Listato 1. La struttura dati utilizza chiavi simboliche per permettere estrazione uniforme mediante Get_LisVal_LisAssoc indipendentemente dal metodo. Calcola automaticamente errore iniziale ed errore finale mediante compute-error, e residui finali mediante compute-residuals. Inverte la lista degli errori (accumulata in ordine inverso durante le iterazioni) per ottenere cronologia corretta. Questa standardizzazione permette alle funzioni di visualizzazione (princ-results-row, princ-best-method, ecc.) di processare uniformemente i risultati dei quattro algoritmi GD, LM, NR, GA. Le chiavi utilizzate sono: 'radius_list, 'start_point, 'point, 'iterations, 'start_error, 'error, 'residuals, 'converged, 'errors_history, 'lambda_final, 'duration, 'name_method.

Valore restituito: Lista associativa con 12 coppie (chiave-simbolica . valore) contenente tutti i dati dell'ottimizzazione.

- **Build_Grid (LisCir dens tipo)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

dens – Numero intero di punti per lato della griglia quadrata (tipicamente potenza di 2: 16, 32, 64, 128, 256)

tipo – Codice numerico della funzione obiettivo da utilizzare (1=MS, 2=EQMR, 3=SQM, 4=ICES)

Scopo: Genera la griglia tridimensionale di punti di errore che costituisce la Fase 1 della pipeline volumetrica nel Listato 2. Calcola il bounding box delle tre circonferenze mediante

Get_BouBox_LisEnt, estrae le coordinate min/max (minX, maxX, minY, maxY), calcola passo griglia (dx, dy) dividendo le dimensioni per (dens-1) per garantire che gli estremi coincidano con i bordi del bounding box. Per ogni coppia (i,j) della griglia (generata mediante range), calcola coordinate cartesiane (Cx, Cy) e valuta la funzione obiettivo mediante Test_Pu_LisCir, negando il risultato prima di assegnarlo come coordinata z per implementare la metafora orografica (minimi errore = massimi superficie = vette). Restituisce matrice bidimensionale di punti 3D organizzata per righe.

Valore restituito: Lista di liste (matrice dens×dens) di punti tridimensionali (x y z), dove z è l'errore negato secondo la funzione obiettivo scelta.

C

- **C:ANALISI ( )**

Parametri: Nessuno (comando interattivo)

Scopo: Funzione comando principale del Listato 2 che implementa l'intera procedura di visualizzazione volumetrica del paesaggio di errore mediante pipeline in cinque fasi coordinate da Sol_Analysis. Orchestra il flusso operativo in sei fasi marcate graficamente da #----#: (1) selezione interattiva tipo analisi (MS/EQMR/SQM/ICES) mediante Get_Tipo_Analisi, determinazione nome tipo per costruzione layer dinamici, stampa conferma scelta; (2) selezione interattiva densità griglia (n=4-8 → $2^n \times 2^n$ punti) mediante Get_Densita_Griglia con tabella informativa, stampa densità calcolata; (3) costruzione nomi layer dinamici mediante concatenazione stringa (Tril_[TIPO]_Superf/Solido/SezCom/SezApi), inizializzazione con colorazione distintiva mediante Lay_Make_Col (40/1/9/6), stampa riepilogo layer utilizzati; (4) selezione interattiva tre circonferenze mediante Get_LisCir; (5) esecuzione pipeline volumetrica completa passando tutti i parametri a Sol_Analysis con flag sezioni attivo; (6) visualizzazione durata totale mediante contatempo. Gestisce backup e ripristino variabili sistema osmode (16384) e 3dosmode (1) per garantire corretta esecuzione comandi nativi AutoCAD BOX, SLICE, SURFNETWORK. Imposta CMDECHO a 0 per sopprimere echo comandi.

Valore restituito: Nessuno (comando di side-effect che genera geometria e stampa output).

- **C:TRILATCOMPARE ( )**

Parametri: Nessuno (comando interattivo)

Scopo: Funzione comando principale del Listato 1 che implementa l'intera procedura comparativa di trilaterazione mediante esecuzione parallela di quattro algoritmi di ottimizzazione. Orchestra il flusso operativo: messaggio introduttivo informativo, selezione interattiva tre circonferenze mediante select-three-circles con validazione tipo e gestione annullamento, inizializzazione quattro layer con colorazione distintiva mediante Lay_Make_Col (TRILAT-GD rosso 1, TRILAT-LM ciano 4, TRILAT-NR magenta 5, TRILAT-GA giallo 6), calcolo punto di partenza come baricentro dei centri delle tre circonferenze mediante Media_lis_Pu applicata ai centri estratti da Get_Data_LisCir, esecuzione sequenziale delle quattro ottimizzazioni optimize-GD, optimize-LM, optimize-NR, optimize-GA (i primi tre ricevono il baricentro come punto di partenza, GA calcola autonomamente il punto iniziale), raccolta quattro strutture risultati in lista, visualizzazione report comparativo completo mediante princ-advanced-results. Gestisce misurazione temporale complessiva mediante doppia chiamata a contatempo (registrazione tempo iniziale e calcolo durata finale). Imposta CMDECHO a 0 per sopprimere echo comandi.

Valore restituito: Nessuno (comando di side-effect che genera geometria e stampa report).

- **calc_grid_err (rect-min rect-max divisions LisCir)**

Parametri:

rect-min – Punto bidimensionale (x y) vertice minimo del rettangolo di ricerca

rect-max – Punto bidimensionale (x y) vertice massimo del rettangolo di ricerca

divisions – Numero intero di divisioni per lato della griglia (punti totali = divisions+1 per lato)

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

Scopo: Valuta la funzione obiettivo SSE/EQMR su una griglia regolare di punti per l'algoritmo GA nel Listato 1. Calcola passo griglia (dx, dy) dividendo dimensioni rettangolo per divisions, genera range coordinate x e y mediante applicazione di lambda su range, estrae centri e raggi delle circonferenze mediante Get_Data_LisCir. Per ogni punto della griglia (doppio ciclo foreach su x-range e y-range), calcola errore mediante compute-error e aggiorna due variabili di tracciamento: min-error (inizializzato a 1e99) con punto Pu_min corrispondente (memorizzando z negativa per coerenza con metafora orografica), max-error (inizializzato a -1e99) con punto Pu_max corrispondente. Utilizzata iterativamente da optimize-GA per identificare la regione più promettente (minimo) e poi raffinare progressivamente la ricerca con griglie più fitte e piccole centrate sul minimo corrente.

Valore restituito: Lista di due punti tridimensionali (Pu_min Pu_max) dove z è l'errore negato, rappresentanti rispettivamente il punto con errore minimo (candidato soluzione) e massimo (per analisi range).

- **compute-error (Pu1 LisCen LisRad)**

Parametri:

Pu1 – Punto tridimensionale candidato da valutare

LisCen – Lista di tre punti (centri delle circonferenze di riferimento)

LisRad – Lista di tre numeri reali positivi (raggi delle circonferenze)

Scopo: Calcola l'errore quadratico medio EQMR (Root Mean Square Error) per un punto candidato rispetto alle tre circonferenze di riferimento nel Listato 1. Implementa la formula EQMR =

sqrt(Σ(di-ri)²/n) dove di è la distanza dal punto al centro i-esimo, ri è il raggio i-esimo, n=3. Converte il punto in 2D mediante Pu_2D prima del calcolo distanze per operare esclusivamente sul piano xy. Eleva al quadrato ciascun residuo (di-ri), somma i tre contributi, divide per 3, estrae radice quadrata. Questa metrica penalizza fortemente gli scarti maggiori ed è sensibile agli outlier, rappresentando la scelta standard per problemi di fitting e trilaterazione. Utilizzata per valutare la qualità delle soluzioni iniziali e finali nelle tabelle comparative, e come funzione obiettivo di riferimento per validazione (equivalente a objective-function con radice e divisione per n).

Valore restituito: Numero reale non negativo rappresentante l'EQMR (unità di misura = unità coordinate disegno).

- **compute-residuals (Pu1 LisCen LisRad)**

Parametri:

Pu1 – Punto tridimensionale da analizzare

LisCen – Lista di tre punti (centri delle circonferenze di riferimento)

LisRad – Lista di tre numeri reali positivi (raggi delle circonferenze)

Scopo: Calcola i tre residui individuali (errori singoli) per un punto rispetto alle tre circonferenze di riferimento nel Listato 1. Per ciascuna circonferenza i-esima, calcola il residuo come (di - ri) dove di è la distanza 2D dal punto al centro i-esimo (calcolata convertendo il punto in 2D mediante Pu_2D) e ri è il raggio i-esimo. I residui possono essere positivi (punto esterno alla circonferenza), negativi (punto interno), o nulli (punto esattamente sulla circonferenza). Utilizzata esclusivamente per l'analisi dettagliata dell'errore nella funzione princ-residuals-by-method che stampa i tre residui separatamente e calcola la media dei valori assoluti, permettendo di visualizzare come l'errore si distribuisce tra le tre misurazioni e identificare eventuali circonferenze problematiche.

Valore restituito: Lista di tre numeri reali (possono essere positivi, negativi o nulli) rappresentanti i residui individuali nell'ordine delle circonferenze.

- **cons_0 (cod)**

Parametri:

cod – Codice numerico intero da 1 a 9 identificante il tipo di entità

Scopo: Restituisce la coppia DXF (0 . "NOME_ENTITA") che definisce il tipo geometrico dell'entità AutoCAD. Mappa codici numerici a nomi stringa mediante cond: 1→POLYLINE, 2→VERTEX, 3→SEQEND, 4→LWPOLYLINE, 5→LINE, 6→CIRCLE, 7→POINT, 8→SPLINE, 9→ELLIPSE. Il codice DXF 0 è obbligatorio come primo elemento della lista DXF e identifica univocamente la classe di entità da creare. Utilizzata in entrambi i listati per costruire liste DXF passate a entmakex per creazione programmatica di entità geometriche. Fornisce astrazione che permette di specificare il tipo entità mediante codice numerico simbolico invece di stringa letterale, facilitando manutenzione e leggibilità del codice. Se il codice non corrisponde a nessun valore mappato, restituisce nil.

Valore restituito: Coppia DXF (0 . "NOME_ENTITA") se codice valido, nil altrimenti.

- **cons_8 (lay)**

Parametri:

lay – Stringa rappresentante il nome del layer di destinazione

Scopo: Restituisce la coppia DXF (8 . "NOME_LAYER") che assegna un'entità al layer specificato. Il codice DXF 8 definisce l'appartenenza layer dell'entità ed è opzionale nella lista DXF (se omesso l'entità viene creata sul layer corrente). Utilizzata in entrambi i listati per costruire liste DXF passate a entmakex per creazione programmatica di entità su layer specifici. Combinata sistematicamente con Lay_Check per garantire che il layer esista prima di assegnare l'entità. Fornisce astrazione che centralizza la costruzione della coppia DXF evitando errori di sintassi nella digitazione manuale del codice 8. Non verifica la validità o esistenza del layer (responsabilità di Lay_Check o Lay_Make_Col).

Valore restituito: Coppia DXF (8 . "NOME_LAYER") dove NOME_LAYER è la stringa parametro ricevuta.

- **cons_10 (Pu1)**

Parametri:

Pu1 – Punto bidimensionale o tridimensionale rappresentante coordinate geometriche

Scopo: Restituisce la coppia DXF (10 . (x y z)) che definisce il primo punto geometrico significativo dell'entità AutoCAD. Il codice DXF 10 identifica: il punto di inserimento per POINT, il centro per CIRCLE, il primo estremo per LINE, il vertice per VERTEX. Garantisce tridimensionalità del punto chiamando Pu_3D che aggiunge z=0.0 se il punto è bidimensionale. Utilizzata nel Listato 1 per creare entità POINT che visualizzano i risultati delle quattro ottimizzazioni sui rispettivi layer. Fornisce astrazione che centralizza la costruzione della coppia DXF gestendo automaticamente la normalizzazione 2D→3D. La tridimensionalità è richiesta da AutoCAD per coerenza anche quando le operazioni sono planari.

Valore restituito: Coppia DXF (10 . (x y z)) dove (x y z) è il punto normalizzato a 3D.

- **cons_11 (Pu1)**

Parametri:

Pu1 – Punto bidimensionale o tridimensionale rappresentante coordinate geometriche

Scopo: Restituisce la coppia DXF (11 . (x y z)) che definisce il secondo punto geometrico significativo o punto direttore dell'entità AutoCAD. Il codice DXF 11 identifica: il secondo estremo per LINE, i punti di controllo per SPLINE (possono essere multipli nella stessa lista DXF). Garantisce tridimensionalità del punto chiamando Pu_3D che aggiunge z=0.0 se il punto è bidimensionale. Utilizzata nel Listato 2 nella funzione Draw_Spline per costruire la lista dei punti di controllo della spline NURBS mediante mapcar

applicata alla lista punti, generando una coppia (11 . punto) per ciascun punto di controllo che viene poi appendicizzata alla lista DXF della spline. La tridimensionalità è necessaria per coerenza geometrica AutoCAD.

Valore restituito: Coppia DXF (11 . (x y z)) dove (x y z) è il punto normalizzato a 3D.

- **cons_100 (cod)**

Parametri:

cod – Codice numerico intero da 1 a 7 identificante la classe ObjectARX

Scopo: Restituisce la coppia DXF (100 . "NOME_CLASSE") che definisce la classe di appartenenza dell'entità nella gerarchia ObjectARX di AutoCAD. Mappa codici numerici a nomi stringa mediante cond: 1→AcDbEntity (classe base generica), 2→AcDb3dPolyline, 3→AcDbVertex, 4→AcDb3dPolylineVertex, 5→AcDbPolyline, 6→AcDbSpline, 7→AcDbEllipse. Il codice DXF 100 può apparire multiple volte nella lista DXF per rappresentare la catena di ereditarietà (es. prima AcDbEntity poi classe specifica). Utilizzata in entrambi i listati per costruire liste DXF passate a entmakex garantendo corretta inizializzazione delle entità secondo il modello ObjectARX. Fornisce astrazione che mappa codici simbolici a stringhe classe evitando errori di digitazione. Se il codice non corrisponde a nessun valore mappato, restituisce nil.

Valore restituito: Coppia DXF (100 . "NOME_CLASSE") se codice valido, nil altrimenti.

- **contatempo (st_ti)**

Parametri:

st_ti – Tempo iniziale precedentemente registrato (numero reale rappresentante DATE AutoCAD), oppure nil per registrare tempo corrente

Scopo: Misura la durata di esecuzione mediante variabile di sistema DATE di AutoCAD (giorni decimali dalla mezzanotte del 1 gennaio 1900). Implementa logica a due fasi: se chiamata con nil, registra e restituisce il tempo corrente come riferimento iniziale; se chiamata con tempo precedente, calcola la differenza moltiplicandola per 86400 (secondi/giorno) per conversione da giorni a secondi, stampa messaggio formattato "Durata esecuzione: X minuti e Y.ZZZ secondi" mediante divisione intera e resto, restituisce la durata numerica in secondi. Utilizzata in entrambi i listati per analisi prestazionale: nel Listato 1 misura sia la durata di ciascuna ottimizzazione singola (GD, LM, NR, GA) sia la durata complessiva del comando; nel Listato 2 misura la durata totale della pipeline volumetrica. La precisione è limitata dalla granularità della variabile DATE (circa millisecondi).

Valore restituito: Alla prima chiamata (con nil): numero reale rappresentante tempo corrente in formato DATE. Alla seconda chiamata (con tempo precedente): numero reale rappresentante durata in secondi.

D

- **determinant-3x3 (A)**

Parametri:

A – Matrice 3×3 rappresentata come lista di tre liste di tre numeri reali ciascuna

Scopo: Calcola il determinante di una matrice 3×3 mediante la regola di Sarrus (sviluppo completo dei sei prodotti: tre positivi e tre negativi). Estrae i nove elementi della matrice mediante mapcar con set su nove variabili simboliche (a11, a12, a13, a21, a22, a23, a31, a32, a33) dopo aver appiattito la matrice con apply 'append. Calcola det(A) = a11·a22·a33 - a11·a23·a32 + a12·a23·a31 - a12·a21·a33 + a13·a21·a32 - a13·a22·a31. Utilizzata nel Listato 1 dall'algoritmo Newton-Raphson per verificare l'invertibilità della matrice Hessiana (se determinante vicino a zero la matrice è singolare) e dalla funzione solve-system-3x3 per applicare la regola di Cramer (calcola quattro determinanti: della matrice coefficienti e di tre matrici con colonna sostituita dal termine noto).

Valore restituito: Numero reale rappresentante il determinante della matrice (può essere positivo, negativo o zero).

- **Diff_2Lis (Lis1 Lis2)**

Parametri:

Lis1 – Lista di numeri (tipicamente coordinate di un punto o componenti di un vettore)

Lis2 – Lista di numeri della stessa lunghezza di Lis1

Scopo: Calcola la differenza elemento per elemento di due liste (operazione vettoriale sottrazione) applicando l'operatore - mediante mapcar. Utilizzata nel Listato 1 per calcolare vettori differenza tra punti nelle funzioni gradient-function e hessian-function (calcolo del vettore diff = point - center necessario per derivate parziali), e per aggiornamenti di posizione negli algoritmi di ottimizzazione. Fornisce astrazione per operazione vettoriale fondamentale senza necessità di cicli espliciti, sfruttando la natura funzionale di AutoLISP. Le liste devono avere la stessa lunghezza altrimenti mapcar si arresta alla lista più corta.

Valore restituito: Lista di numeri della stessa lunghezza delle liste parametro, contenente le differenze elemento per elemento (Lis1[i] - Lis2[i]).

- **Draw_Box_LisPu (LisPu lay)**

Parametri:

LisPu – Lista di due punti tridimensionali (vertici opposti del parallelepipedo)

lay – Stringa rappresentante il nome del layer di destinazione

Scopo: Crea un solido 3D di tipo parallelepipedo (box) definito da due vertici opposti mediante comando nativo AutoCAD BOX. Gestisce il layer corrente salvando il valore precedente, verificando/creando il layer di destinazione mediante Lay_Check, impostando il layer corrente con setvar "clayer", eseguendo il comando BOX con opzioni non-interattive "_non" per sopprimere

snap, ripristinando il layer originale. Restituisce l'entità creata recuperandola con entlast. Utilizzata nel Listato 2 nella Fase 4 della pipeline volumetrica dalla funzione Draw_Sol_From_Surnet per creare il parallelepipedo di base che verrà poi "affettato" dalla superficie mediante SLICE, generando il solido finale con "tetto" rappresentante il paesaggio di errore.

Valore restituito: ENAME dell'entità 3DSOLID appena creata.

- **Draw_Point (Pu1 lay)**

Parametri:

Pu1 – Punto bidimensionale o tridimensionale da disegnare

lay – Stringa rappresentante il nome del layer di destinazione

Scopo: Crea un'entità di tipo POINT alle coordinate specificate sul layer indicato mediante costruzione programmatica di lista DXF passata a entmakex. Verifica preventivamente che Pu1 sia un punto valido mediante predicato P_Pu, verifica/crea il layer mediante Lay_Check, costruisce lista DXF assemblando le coppie generate da cons_0 (tipo POINT), cons_100 (classe AcDbEntity), cons_10 (coordinate normalizzate 3D), cons_8 (layer). Utilizzata nel Listato 1 dalle quattro funzioni di ottimizzazione optimize-GD, optimize-LM, optimize-NR, optimize-GA per visualizzare graficamente i punti risultanti sui rispettivi layer con colorazione distintiva (rosso, ciano, magenta, giallo), permettendo confronto visivo immediato delle soluzioni calcolate dai diversi algoritmi.

Valore restituito: ENAME dell'entità POINT appena creata, oppure nil se Pu1 non è un punto valido.

- **Draw_Sections_From_Solid (sol_ent LisCz num_sez lay)**

Parametri:

sol_ent – ENAME di un'entità 3DSOLID da sezionare

LisCz – Lista di due numeri reali (minZ maxZ) definenti il range verticale per le sezioni

num_sez – Numero intero di sezioni da generare (equidistanti nel range)

lay – Stringa rappresentante il nome del layer di destinazione per le curve di livello

Scopo: Genera curve di livello (sezioni orizzontali) di un solido 3D mediante sezionamento con piani paralleli equidistanti, utilizzando il metodo ActiveX SectionSolid su oggetto VLA. Fase 5 della pipeline volumetrica nel Listato 2. Gestisce il layer corrente salvando, verificando/creando, impostando e ripristinando. Converte l'ENAME in oggetto VLA mediante vlax-ename->vla-object. Calcola passo verticale stepZ = (maxZ-minZ)/num_sez. Itera da 0 a num_sez inclusi calcolando quota z = minZ + i·stepZ per ciascuna sezione. Per ogni quota chiama vla-SectionSolid passando tre punti non collineari sul piano z (origine, punto su asse x, punto su asse y) convertiti in VLA-POINT 3D mediante vlax-3d-point. Le curve risultanti vengono create automaticamente come entità REGION sul layer corrente. Utilizzata due volte da Sol_Analysis: una per sezioni complessive (passo maggiore), una per sezioni apicali (passo minore) per visualizzazione dettagliata della zona critica vicino al minimo.

Valore restituito: Nessuno (funzione di side-effect che genera entità REGION nel disegno).

- **Draw_Sol_From_Surnet (surf_Ent LisPuBox lay)**

Parametri:

surf_Ent – ENAME di un'entità SURFACE (superficie loftata NURBS)

LisPuBox – Lista di due punti tridimensionali (vertici opposti del bounding box della griglia)

lay – Stringa rappresentante il nome del layer di destinazione

Scopo: Costruisce il solido volumetrico rappresentante il paesaggio di errore mediante operazione booleana di slice (affettamento). Fase 4 della pipeline volumetrica nel Listato 2. Estrae i due punti del bounding box, crea un parallelepipedo di base con altezza zero (terzo punto ha z=0) mediante Draw_Box_LisPu che funge da "massa" da tagliare. Salva e imposta variabili sistema osmode (16384, disabilita tutti gli snap) e 3dosmode (1, abilita snap 3D) per corretta esecuzione comando SLICE. Esegue comando nativo ._SLICE passando il box, nessuna opzione taglio aggiuntiva (""), modalità "_Surface" (usa superficie come lama), la superficie, opzione "_B" (mantieni entrambe le parti). Il comando genera due solidi: uno sotto e uno sopra la superficie. Recupera entrambi gli ENAME (box_Ent1 è l'originale, box_Ent2 è l'ultimo creato), estrae le quote z dei loro centroidi mediante proprietà ActiveX Centroid, confronta le quote ed elimina il solido con centroide più basso mantenendo quello con "tetto" della superficie di errore. Ripristina variabili sistema.

Valore restituito: ENAME del 3DSOLID mantenuto (quello con centroide più alto), oppure nil se operazione fallisce.

- **Draw_Spl_From_Grid (rmse_grid lay grado)**

Parametri:

rmse_grid – Matrice bidimensionale (lista di liste) di punti tridimensionali rappresentanti la griglia di errore

lay – Stringa rappresentante il nome del layer di destinazione

grado – Numero intero specificante il grado della spline NURBS (1=spezzata, 2=quadratica, 3=cubica)

Scopo: Genera due famiglie ortogonali di spline (per righe e per colonne della griglia) mediante interpolazione dei punti tridimensionali. Fase 2 della pipeline volumetrica nel Listato 2. Inizializza due liste accumulatori (x_splines per righe, y_splines per colonne). Per ogni riga della matrice, chiama Draw_Spline passando la lista di punti della riga, il layer, chiusura disabilitata (nil), e grado, accumulando gli ENAME risultanti in x_splines (inverte per preservare ordine). Per le colonne, itera sugli indici (0 fino a lunghezza prima riga) e per ciascun indice estrae la colonna corrispondente mediante foreach con append costruendo lista verticale, poi chiama Draw_Spline accumulando in y_splines. Inverte entrambe le liste per ordine corretto. Queste spline formano

una rete di sezioni che definisce la forma della superficie da costruire con SURFNETWORK nella fase successiva.

Valore restituito: Lista di due elementi: (x_splines y_splines) dove ciascuno è una lista di ENAME delle spline create.

- **Draw_Spline (lPu lay clo grado)**

Parametri:

lPu – Lista di punti tridimensionali (punti di controllo della spline)

lay – Stringa rappresentante il nome del layer di destinazione

clo – Booleano (T per spline chiusa, nil per spline aperta)

grado – Numero intero specificante il grado della spline NURBS (1=lineare/spezzata, 2=quadratica, 3=cubica, fino a n-1 dove n è il numero di punti)

Scopo: Crea una spline NURBS parametrica mediante costruzione programmatica di lista DXF passata a entmakex. Verifica preventivamente che ci siano almeno 2 punti e che il loro numero sia maggiore del grado richiesto (altrimenti la spline non è definita). Verifica/crea il layer mediante Lay_Check. Costruisce lista DXF assemblando: cons_0 (tipo SPLINE), due cons_100 (classi AcDbEntity e AcDbSpline), coppia (70 . flag) dove flag è 1 se chiusa 0 altrimenti, cons_8 (layer), coppia (71 . grado), coppia (72 . numero-punti), seguita da sequenza di coppie (11 . punto) generate mediante mapcar con lambda che applica cons 11 a ciascun punto della lista. Utilizzata nel Listato 2 dalla funzione Draw_Spl_From_Grid con grado 3 (spline cubica) per interpolare i punti della griglia di errore generando curve lisce che definiscono la superficie.

Valore restituito: ENAME dell'entità SPLINE appena creata, oppure nil se condizioni non soddisfatte.

- **Draw_SurNet_From_Spl (spline_lists lay del_flag)**

Parametri:

spline_lists – Lista di due elementi (xlist ylist) dove ciascuno è una lista di ENAME di spline (famiglia orizzontale e verticale)

lay – Stringa rappresentante il nome del layer di destinazione per la superficie

del_flag – Booleano (T per eliminare le spline dopo creazione superficie, nil per mantenerle)

Scopo: Crea una superficie loftata NURBS complessa dalla rete di spline mediante comando nativo AutoCAD SURFNETWORK. Fase 3 della pipeline volumetrica nel Listato 2. Gestisce il layer corrente salvando, verificando/creando mediante Lay_Check, impostando con setvar "clayer". Estrae le due liste di ENAME (xlist per righe, ylist per colonne). Costruisce dinamicamente la lista di argomenti per il comando assemblando: stringa comando "._surfnetwork", tutti gli ENAME di xlist, stringa vuota "" (termina prima famiglia), tutti gli ENAME di ylist, stringa vuota "" (termina seconda famiglia). Applica command-s (versione sincrona safe del comando che previene interruzioni) alla lista assemblata con append. Recupera l'ENAME della superficie creata con entlast. Se del_flag è T, elimina tutte le spline di entrambe le famiglie mediante foreach con Ent_Erase per pulizia grafica (le spline sono intermedie, interessa solo la superficie). Ripristina layer originale.

Valore restituito: ENAME dell'entità SURFACE appena creata.

E

- **EQMR (lst len)**

Parametri:

lst – Lista di numeri reali (tipicamente distanze o errori)

len – Numero intero rappresentante la lunghezza della lista (numero di elementi)

Scopo: Calcola la Radice quadrata dell'Errore Quadratico Medio (Root Mean Square Error, RMSE) di una lista di valori. Implementa la formula EQMR = sqrt(Σ(xi²)/n). Eleva al quadrato ciascun elemento della lista mediante mapcar con lambda (lambda (d) (* d d)), somma i quadrati con apply '+, divide per len, estrae radice quadrata con sqrt. Seconda delle quattro metriche statistiche disponibili nel Listato 2. Questa funzione obiettivo penalizza fortemente gli scarti maggiori (per l'elevazione al quadrato) ed è molto sensibile agli outlier. Quando usata per mappare il paesaggio di errore genera superfici con morfologia relativamente semplice (vulcano, catena montuosa, fondovalle) a seconda della configurazione geometrica delle circonferenze. Utilizzata sia direttamente come metrica selezionabile (tipo=2) sia come componente della metrica ibrida ICES.

Valore restituito: Numero reale non negativo rappresentante l'EQMR (unità di misura = unità coordinate disegno).

- **Ent_Erase (ent)**

Parametri:

ent – ENAME di un'entità AutoCAD da eliminare

Scopo: Elimina dal database del disegno AutoCAD l'entità specificata mediante comando nativo entdel. Verifica preventivamente che il parametro sia effettivamente di tipo ENAME mediante predicato (= (type ent) 'ENAME) per evitare errori se viene passato nil o un tipo diverso. Funzione di utilità per la pulizia di entità temporanee o indesiderate utilizzata in entrambi i listati. Nel Listato 2 è utilizzata da Draw_SurNet_From_Spl per eliminare le spline intermedie dopo creazione della superficie (se flag attivo) e da Draw_Sol_From_Surnet per eliminare uno dei due solidi risultanti dall'operazione SLICE mantenendo solo quello con centroide più alto. Fornisce astrazione che centralizza la verifica tipo e l'operazione di eliminazione. L'entità non viene fisicamente cancellata ma marcata come "deleted" nel database; può essere ripristinata con entdel richiamata sulla stessa entità.

Valore restituito: ENAME dell'entità eliminata se operazione riuscita, nil altrimenti.

F

- **format-point (Pu1 3D)**

Parametri:

Pu1 – Punto bidimensionale o tridimensionale da formattare

3D – Booleano o valore non-nil (T per includere coordinata z, nil per formato 2D)

Scopo: Formatta un punto come stringa leggibile nel formato "(x, y)" oppure "(x, y, z)" con precisione controllata e padding per allineamento. Utilizzata nel Listato 1 per visualizzare coordinate nelle tabelle comparative di output. Chiama rtos-pad per ciascuna coordinata specificando modalità 2 (decimale), precisione 6 decimali per x e y (8 per z), larghezza totale 4 caratteri per parte intera di x e y (8 per z). Concatena le stringhe formattate inserendo parentesi tonde, virgole e spazi come separatori. Se parametro 3D è non-nil aggiunge la terza coordinata preceduta da virgola e spazio. La funzione rtos-pad gestisce automaticamente il padding degli zeri decimali e degli spazi per la parte intera garantendo allineamento colonnare quando multiple coordinate vengono stampate verticalmente.

Valore restituito: Stringa nel formato "(x.xxxxxx , y.yyyyyy)" o "(x.xxxxxx , y.yyyyyy , z.zzzzzzzz)" con padding per allineamento.

G

- **Get_BouBox_Ent (ent)**

Parametri:

ent – ENAME di un'entità AutoCAD

Scopo: Calcola il bounding box (rettangolo di contenimento 3D) di un'entità mediante metodo ActiveX GetBoundingBox applicato all'oggetto VLA corrispondente. Verifica che ent sia non-nil, converte l'ENAME in oggetto VLA mediante vlax-ename->vla-object, invoca vla-GetBoundingBox passando due simboli atomici (Pu_min e Pu_max) che verranno associati a safearray contenenti le coordinate dei vertici opposti del box. Converte i due safearray in liste AutoLISP normali mediante vlax-safearray->list, applica LisPuBox_LisPu sui due punti (che non fa altro che restituire la lista dei due punti già ricevuti, forse ridondante). Utilizzata nel Listato 2 per calcolare il bounding box di singole circonferenze, necessario per determinare la regione di campionamento nella generazione della griglia di errore.

Valore restituito: Lista di due punti tridimensionali (Pu_min Pu_max) rappresentanti vertici opposti del bounding box, oppure nil se ent è nil.

- **Get_BouBox_LisEnt (LisEnt)**

Parametri:

LisEnt – Lista di ENAME di entità AutoCAD

Scopo: Calcola il bounding box complessivo (unione) di una lista di entità applicando Get_BouBox_Ent a ciascuna entità e calcolando il rettangolo che contiene tutti i bounding box individuali. Verifica che LisEnt sia non-nil, applica mapcar 'Get_BouBox_Ent ottenendo lista di coppie (Pu_min Pu_max) per ciascuna entità, appiattisce la struttura con apply 'append ottenendo lista piatta di punti, applica LisPuBox_LisPu che calcola il punto minimo globale (coordinate minime su ciascun asse) e punto massimo globale (coordinate massime su ciascun asse). Utilizzata nel Listato 2 dalla funzione Build_Grid per determinare il dominio della griglia di errore: calcola il bounding box delle tre circonferenze di riferimento garantendo che la griglia le racchiuda tutte, definendo così la regione candidata per la visualizzazione del paesaggio di errore.

Valore restituito: Lista di due punti tridimensionali (Pu_min_globale Pu_max_globale) rappresentanti vertici opposti del bounding box complessivo, oppure nil se LisEnt è nil.

- **Get_Data_Cir (ent)**

Parametri:

ent – ENAME di un'entità AutoCAD di tipo CIRCLE

Scopo: Estrae i dati geometrici caratteristici di un cerchio (centro e raggio) dalla sua entity list DXF. Verifica che ent sia un ENAME valido mediante predicato P_ent, recupera la entity list completa con entget, estrae i valori corrispondenti ai codici DXF 10 (centro, punto 3D) e 40 (raggio, numero reale) mediante Get_LisVal_LisAssoc passando la lista dei due codici. Utilizzata nel Listato 1 per estrarre dati dalle tre circonferenze selezionate dall'utente per il problema di trilaterazione. Fornisce astrazione che centralizza l'accesso ai dati geometrici specifici dei cerchi evitando chiamate multiple a assoc e gestendo automaticamente la validazione del parametro. Se ent è nil o non valido, restituisce nil.

Valore restituito: Lista di due elementi (centro raggio) dove centro è punto 3D (x y z) e raggio è numero reale, oppure nil se ent non valido.

- **Get_Data_LisCir (LisCir)**

Parametri:

LisCir – Lista di ENAME di entità AutoCAD di tipo CIRCLE

Scopo: Estrae i dati geometrici da una lista di cerchi e riorganizza i risultati in due liste separate (centri e raggi). Applica mapcar 'Get_Data_Cir alla lista di ENAME ottenendo lista di coppie (centro raggio), poi applica apply 'mapcar (cons 'list ...) che è un pattern idiomatico AutoLISP per trasporre una matrice: converte lista di coppie [(c1 r1) (c2 r2) (c3 r3)] in coppia di liste ([c1 c2 c3] [r1 r2 r3]). Utilizzata nel Listato 1 da tutte le funzioni di ottimizzazione per organizzare i dati delle tre circonferenze di riferimento in un formato comodo per i calcoli successivi: gli algoritmi iterano sugli indici 0-2 accedendo ai centri e raggi mediante nth. Fornisce astrazione che separa la fase di estrazione dati dalla fase di elaborazione algoritmica.

Valore restituito: Lista di due elementi (LisCen LisRad) dove LisCen è lista di centri 3D e LisRad è lista di raggi reali.

- **Get_Densita_Griglia ( )**

Parametri: Nessuno (funzione interattiva)

Scopo: Gestisce la selezione interattiva della densità della griglia per il Listato 2 presentando una tabella informativa ASCII-art e validando l'input. Stampa tabella con 5 righe (n=4-8) mostrando per ciascun valore: n (esponente), densità risultante ($2^n \times 2^n$), punti totali griglia (2^{2n}). Entra in ciclo while richiedendo input mediante getint con prompt che include default 6. Se utente preme Invio senza digitare (val è nil), imposta val a 6. Valida che val sia membro della lista (4 5 6 7 8), altrimenti stampa messaggio errore e ripete richiesta. Esce dal ciclo solo quando val è valido. Utilizzata nella Fase 2 del comando C:ANALISI. La tabella informativa aiuta l'utente a comprendere l'impatto della scelta sulla dimensione computazionale: da 16×16=256 punti (n=4, veloce) a 256×256=65.536 punti (n=8, molto lento ma dettagliato). Default 6 (64×64=4.096 punti) è equilibrio ragionevole.

Valore restituito: Numero intero validato tra 4 e 8 (esponente per calcolo densità 2^n).

- **Get_DxfVal_Ent (ent code)**

Parametri:

ent – ENAME di un'entità AutoCAD

code – Codice numerico DXF intero (es. 0=tipo, 8=layer, 10=punto, 40=raggio)

Scopo: Estrae il valore di un singolo attributo DXF specifico da un'entità dato il codice numerico. Verifica che ent sia un ENAME valido mediante predicato P_ent, recupera la entity list completa con entget, chiama Get_LisVal_LisAssoc passando la entity list e una lista contenente il singolo code, estrae il primo (e unico) elemento della lista risultante con car. Wrapper semplificato per accesso agli attributi DXF utilizzato pervasivamente in entrambi i listati quando serve estrarre un solo valore specifico (es. verificare tipo entità con codice 0). Fornisce astrazione che nasconde la complessità della entity list e della ricerca con assoc, gestendo automaticamente la validazione del parametro. Se ent è nil o code non presente, restituisce nil.

Valore restituito: Valore associato al codice DXF (tipo dipende dal codice: stringa per 0/8, punto per 10, numero per 40), oppure nil se non trovato.

- **Get_LisCir (NumCir)**

Parametri:

NumCir – Numero intero di circonferenze da selezionare

Scopo: Gestisce la selezione interattiva guidata di un numero parametrico di circonferenze con validazione tipo e gestione annullamento. Versione generalizzata usata nel Listato 2 (chiamata con NumCir=3). Inizializza variabili LisCir (accumulatore), ent (entità corrente), done (flag terminazione). Entra in ciclo while che continua finché non terminato e lista non completa. Per ogni iterazione, richiede selezione mediante entsel con prompt dinamico che indica "cerchio N di NumCir", estrae ENAME con car. Gestisce tre casi mediante cond: (1) se ent è nil (utente ha annullato con ESC/Enter), verifica se lista parzialmente popolata (mostra messaggio annullamento, azzera lista, termina) oppure termina silenziosamente; (2) se ent non nil, verifica tipo mediante Get_DxfVal_Ent con codice 0, se è "CIRCLE" accumula in lista e stampa feedback con ENAME, altrimenti stampa messaggio errore. Esce dal ciclo restituendo lista completa o nil se annullato.

Valore restituito: Lista di NumCir ENAME di entità CIRCLE se selezione completata, nil se annullata.

- **Get_LisVal_LisAssoc (LisAssoc Liskey)**

Parametri:

LisAssoc – Lista associativa (lista di coppie chiave-valore DXF o simboliche)

Liskey – Lista di chiavi (codici numerici DXF o simboli) da estrarre

Scopo: Estrae i valori corrispondenti a una lista di chiavi da una lista associativa in una sola operazione. Applica mapcar con lambda che per ciascuna chiave della Liskey esegue assoc sulla LisAssoc e estrae il valore con cdr. Funzione di utilità per estrarre multipli attributi DXF o elementi di strutture dati in una sola chiamata, evitando ripetizioni di codice. Utilizzata pervasivamente in entrambi i listati: per estrarre dati geometrici da entity list (centri e raggi dei cerchi), per estrarre elementi da strutture risultati delle ottimizzazioni (punti, errori, convergenza, durata), per decomposizione compatta di strutture dati complesse mediante mapcar 'set che associa simboli a valori estratti. L'ordine dei valori restituiti corrisponde all'ordine delle chiavi in Liskey.

Valore restituito: Lista di valori corrispondenti alle chiavi nell'ordine di Liskey, con nil per chiavi non presenti in LisAssoc.

- **Get_Tipo_Analisi ( )**

Parametri: Nessuno (funzione interattiva)

Scopo: Gestisce la selezione interattiva del tipo di funzione obiettivo per il Listato 2 mediante keyword vincolate. Chiama initget con stringa "MS EQMR SQM ICES" che definisce le opzioni valide (case-insensitive), poi chiama getkword con prompt che include default EQMR tra parentesi angolari. L'utente può digitare keyword completa o abbreviazione univoca. Mappa il risultato a codice numerico mediante cond: "MS"→1 (Media Semplice), "EQMR"→2 (Root Mean Square Error), "SQM"→3 (Scarto Quadratico Medio), "ICES"→4 (Indice Composto Errore-Scarto), nil→2 (se utente preme Invio accetta default), T→2 (fallback se valore inatteso). Utilizzata nella Fase 1 del comando C:ANALISI per determinare quale metrica statistica utilizzare nella mappatura del paesaggio di errore, influenzando direttamente la morfologia della superficie risultante.

Valore restituito: Numero intero da 1 a 4 rappresentante il tipo di analisi selezionato (o default 2 se Invio).

- **gradient-function (point LisCen LisRad)**

Parametri:

point – Punto tridimensionale candidato nel quale calcolare il gradiente

LisCen – Lista di tre punti (centri delle circonferenze di riferimento)

LisRad – Lista di tre numeri reali positivi (raggi delle circonferenze)

Scopo: Calcola il vettore gradiente (derivate parziali prime) della funzione obiettivo SSE nel punto specificato per gli algoritmi di ottimizzazione del Listato 1. Inizializza vettore gradiente a (0 0 0). Per ciascuna delle tre circonferenze (iterazione su indici 0-2 con foreach su range), calcola vettore differenza diff = point - center[i] mediante Diff_2Lis, calcola norma (distanza) dist mediante vector-length, gestisce singolarità verificando che dist > 1e-10 per evitare divisione per zero quando il punto coincide con un centro. Se dist sufficiente, calcola termine scalare = 2·(dist - radius[i]), calcola contributo vettoriale grad-term = diff·(term/dist) mediante vector-scale, accumula il contributo nel gradiente totale mediante Somm_2Lis. Il gradiente punta nella direzione di massima crescita della funzione obiettivo; gli algoritmi GD, NR, LM lo usano negato per muoversi verso il minimo. Utilizzata ad ogni iterazione degli algoritmi per determinare direzione di discesa ottimale.

Valore restituito: Vettore tridimensionale (lista di tre numeri reali) rappresentante il gradiente ∇f(point) = (∂f/∂x, ∂f/∂y, ∂f/∂z).

H

- **hessian-function (point LisCen LisRad)**

Parametri:

point – Punto tridimensionale candidato nel quale calcolare la Hessiana

LisCen – Lista di tre punti (centri delle circonferenze di riferimento)

LisRad – Lista di tre numeri reali positivi (raggi delle circonferenze)

Scopo: Calcola la matrice Hessiana 3×3 (derivate parziali seconde) della funzione obiettivo SSE nel punto specificato per gli algoritmi NR e LM del Listato 1. Inizializza matrice H a zero (tre liste di tre zeri). Per ciascuna delle tre circonferenze (iterazione su indici 0-2), calcola vettore differenza diff = point - center[i], distanza dist, gestisce singolarità verificando dist > 1e-10. Se dist sufficiente, calcola due termini: term1 = 2·(dist-radius[i])/dist (contributo lineare), term2 = 2·radius[i]/dist³ (contributo curvatura). Per ogni coppia di indici (j,k) della matrice 3×3 (doppio ciclo su 0-2), estrae riga j corrente, calcola elemento H[j][k]: se j=k (elemento diagonale) somma term1 + term2·diff[j]², altrimenti (elemento extra-diagonale) somma term2·diff[j]·diff[k]. Aggiorna elemento mediante subst (sostituisce valore vecchio con nuovo nella riga), aggiorna riga nella matrice mediante altro subst. La Hessiana descrive la curvatura locale della funzione permettendo approssimazione quadratica per convergenza rapida.

Valore restituito: Matrice 3×3 (lista di tre liste di tre numeri reali) rappresentante la Hessiana H(point) dove H[i][j] = ∂²f/∂xi∂xj.

I

- **ICES (lst len)**

Parametri:

lst – Lista di numeri reali (tipicamente distanze o errori)

len – Numero intero rappresentante la lunghezza della lista

Scopo: Calcola l'Indice Composto tra Errore e Scarto come media geometrica di EQMR e SQM. Implementa la formula ICES = sqrt(EQMR·SQM). Chiama entrambe le funzioni EQMR e SQM passando lista e lunghezza, moltiplica i due risultati, estrae radice quadrata. Quarta metrica statistica del Listato 2. Metrica ibrida originale che bilancia magnitudo dell'errore (EQMR) e coerenza/dispersione (SQM), fornendo una valutazione bilanciata. Quando usata per mappare il paesaggio di errore genera superfici articolate con molteplici vette di altezza comparabile evidenziando regioni che rappresentano buon compromesso tra basso errore medio e alta coerenza. La struttura intrinseca della funzione è collegata al problema di Apollonio (CCC) perché la componente SQM trova naturalmente i punti di equidistanza (dove SQM=0), mentre EQMR seleziona tra questi quello con minima somma distanze, guidando verso la soluzione geometricamente ottimale.

Valore restituito: Numero reale non negativo rappresentante l'ICES (unità di misura = unità coordinate disegno).

- **itoa-pad (num width)**

Parametri:

num – Numero intero da convertire in stringa

width – Numero intero rappresentante la larghezza totale desiderata della stringa risultante

Scopo: Converte un numero intero in stringa con padding di spazi a sinistra per raggiungere larghezza specificata garantendo allineamento colonnare. Converte num in stringa mediante itoa, calcola lunghezza stringa risultante con strlen, calcola padding necessario come width - strlen, genera stringa di spazi mediante make-spaces (se padding negativo genera stringa vuota), concatena padding e numero con strcat. Utilizzata nel Listato 1 dalla funzione princ-methods-with-convergence per allineare i numeri di iterazioni nella tabella comparativa (tipicamente con width=4 per gestire iterazioni fino a 9999). Fornisce astrazione per formattazione numerica intera che non è disponibile nativamente in AutoLISP (a differenza di rtos per reali che supporta padding ma solo decimale).

Valore restituito: Stringa contenente spazi di padding seguiti dalla rappresentazione decimale del numero intero, lunghezza totale = max(width, lunghezza-originale).

L

- **Lay_Check (lay)**

Parametri:

lay – Stringa rappresentante il nome del layer da verificare/creare

Scopo: Verifica l'esistenza di un layer nel disegno e lo crea automaticamente con colore di default se non esiste. Funzione di sicurezza utilizzata prima di disegnare entità su layer specifici per garantire che il layer di destinazione sia sempre presente. Chiama P_Layer per verificare esistenza mediante ricerca nella tabella simboli, se restituisce nil (layer non esiste) chiama Lay_Make_Col passando il nome layer e colore 7 (bianco/nero) come default. Utilizzata pervasivamente in entrambi i listati prima di operazioni di disegno: da Draw_Point, Draw_Spline, Draw_Box_LisPu, Draw_Sections_From_Solid, Draw_SurNet_From_Spl per garantire che le entità create non falliscano per layer inesistente. Fornisce astrazione che separa la logica di verifica/creazione layer dalla logica di disegno, semplificando le funzioni chiamanti che non devono preoccuparsi della gestione layer.

Valore restituito: Nessuno (funzione di side-effect che può creare layer se necessario).

- **Lay_Make_Col (lay col)**

Parametri:

lay – Stringa rappresentante il nome del layer da creare/modificare

col – Numero intero rappresentante il codice colore AutoCAD (1-255), oppure nil o <=0 per default bianco (7)

Scopo: Crea un nuovo layer con colore specificato se non esiste, oppure modifica il colore di un layer esistente. Utilizzata per inizializzare i layer di output con colorazione distintiva in entrambi i listati. Verifica esistenza layer mediante P_Layer, normalizza il colore (se nil o <=0 usa 7), esegue comando nativo ._-layer con opzioni appropriate: se layer non esiste usa "_m" (make, crea e imposta come corrente) seguito da "_c" (color) e colore; se layer esiste usa solo "c" per modificare colore. Il doppio passaggio del nome layer nel comando è necessario per sintassi comando nativo. Restituisce il nome layer. Utilizzata nel Listato 1 per creare TRILAT-GD/LM/NR/GA con colori 1/4/5/6, nel Listato 2 per creare layer dinamici Tril[TIPO]_Superf/Solido/SezCom/SezApi con colori 40/1/9/6.

Valore restituito: Stringa rappresentante il nome del layer creato/modificato (parametro lay ricevuto).

- **LisDec_Num (num)**

Parametri:

num – Numero reale o intero di cui calcolare le potenze di 10

Scopo: Genera la lista delle potenze di 10 dalla unità fino all'ordine di grandezza superiore al numero assoluto specificato mediante iterazione moltiplicativa. Inizializza variabile val a 1 e lista accumulatore out a nil. Calcola valore assoluto del numero. Itera con while finché val <= num: accumula val corrente in out mediante cons, moltiplica val per 10. Esce dal ciclo quando val supera num, aggiunge comunque val finale (primo ordine superiore) alla lista, inverte lista con reverse per ordine crescente. Ad esempio, per num=3456 genera [1, 10, 100, 1000, 10000]. Utilizzata nel Listato 2 dalla funzione OrdineDimensione per analizzare l'ordine di grandezza dei valori z del paesaggio di errore e determinare automaticamente il passo ottimale delle sezioni orizzontali nella Fase 5 della pipeline volumetrica.

Valore restituito: Lista di numeri interi rappresentanti potenze consecutive di 10 (da 1 fino alla prima potenza > |num|).

- **LisPuBox_LisPu (LisPu)**

Parametri:

LisPu – Lista di punti tridimensionali

Scopo: Calcola il bounding box (rettangolo di contenimento 3D) di una lista di punti restituendo lista di due punti rappresentanti vertici opposti (minimo e massimo). Applica PuMin_LisPu per calcolare punto con coordinate minime su ciascun asse (x_min, y_min, z_min) e PuMax_LisPu per punto con coordinate massime (x_max, y_max, z_max), assembla i due punti in lista. Utilizzata in entrambi i listati per definire regioni di ricerca e domini di campionamento. Nel Listato 1 è utilizzata da Rect_Ini per determinare il rettangolo iniziale dell'algoritmo GA campionando i punti cardinali delle tre circonferenze, e da Get_BouBox_Ent/Get_BouBox_LisEnt come step finale della conversione safearray→lista. Nel Listato 2 è utilizzata da Build_Grid per determinare il dominio della griglia di errore e da Get_BouBox_LisEnt per calcolare bounding box complessivo di entità.

Valore restituito: Lista di due punti tridimensionali (Pu_min Pu_max) dove Pu_min ha coordinate minime e Pu_max ha coordinate massime su ciascun asse.

- **LisPuQuad_Cir (cir)**

Parametri:

cir – ENAME di un'entità AutoCAD di tipo CIRCLE

Scopo: Genera i quattro punti cardinali (0°, 90°, 180°, 270°) sulla circonferenza di un cerchio per campionamento geometrico. Estrae centro e raggio del cerchio mediante Get_Data_Cir associando i due valori alle variabili cen e rad con mapcar 'set. Applica funzione polar (calcola punto dato centro, angolo e distanza) a quattro angoli: 0 (est), π (ovest), π/2 (nord), 3π/2 (sud), utilizzando il raggio come distanza. Genera lista di quattro punti ordinati in senso antiorario partendo da est. Utilizzata nel Listato 1 esclusivamente dalla funzione Rect_Ini che la applica alle tre circonferenze di riferimento mediante mapcar, genera 12 punti totali (4 per cerchio), li appiattisce con apply 'append, calcola il loro bounding box per determinare il rettangolo iniziale di ricerca dell'algoritmo GA. Questo approccio garantisce che la regione iniziale racchiuda completamente tutte e tre le circonferenze.

Valore restituito: Lista di quattro punti tridimensionali sulla circonferenza negli orientamenti cardinali.

M

- **make-spaces (i)**

Parametri:

i – Numero intero di spazi da generare

Scopo: Genera una stringa contenente esattamente i spazi mediante iterazione e concatenazione. Inizializza stringa vuota, itera i volte con repeat concatenando uno spazio per ogni iterazione mediante strcat, restituisce stringa risultante. Utilizzata nel Listato 1 come funzione di supporto per le funzioni di formattazione itoa-pad e rtos-pad che necessitano padding di spazi per allineamento colonnare nelle tabelle di output. Anche utilizzata dalla funzione --- per costruire margini delle righe di separazione (sebbene questa generi trattini, non spazi). Fornisce astrazione per generazione stringhe ripetitive evitando costruzione manuale con concatenazioni multiple. Se i <= 0 restituisce stringa vuota.

Valore restituito: Stringa contenente i caratteri spazio, oppure stringa vuota se i <= 0.

- **make-zeros (i)**

Parametri:

i – Numero intero di zeri da generare

Scopo: Genera una stringa contenente esattamente i caratteri "0" mediante iterazione e concatenazione. Inizializza stringa vuota, itera i volte con repeat concatenando uno zero per ogni iterazione mediante strcat, restituisce stringa risultante. Utilizzata nel Listato 1 esclusivamente dalla funzione rtos-pad per completare la parte decimale dei numeri formattati quando la precisione richiesta è maggiore della precisione naturale del numero (es. 3.5 con precisione 3 diventa "3.500"). Fornisce astrazione per padding decimale garantendo che tutti i numeri in una colonna abbiano lo stesso numero di decimali visibili per perfetto allineamento verticale. Se i <= 0 restituisce stringa vuota.

Valore restituito: Stringa contenente i caratteri "0", oppure stringa vuota se i <= 0.

- **matrix-identity (dim)**

Parametri:

dim – Numero intero rappresentante la dimensione della matrice quadrata (numero righe = numero colonne)

Scopo: Genera una matrice identità quadrata di dimensione specificata (matrice con 1.0 sulla diagonale principale e 0.0 altrove). Applica mapcar su range 0..(dim-1) generato da range, per ogni indice riga i applica lambda che a sua volta applica mapcar su range 0..(dim-1) per gli indici colonna j, per ogni coppia (i,j) restituisce 1.0 se i=j (elemento diagonale) altrimenti 0.0. Risultato è lista di dim liste di dim elementi ciascuna. Utilizzata nel Listato 1 esclusivamente dall'algoritmo Levenberg-Marquardt nella costruzione della matrice Hessiana modificata H+λI dove I è la matrice identità 3×3 scalata per il parametro di smorzamento λ. L'aggiunta di λI alla Hessiana regolarizza il sistema prevenendo singolarità e interpolando tra comportamento Gradient Descent (λ grande) e Newton-Raphson (λ piccolo).

Valore restituito: Matrice dim×dim (lista di dim liste di dim numeri reali) con 1.0 sulla diagonale e 0.0 altrove.

- **matrix-scale (m scalar)**

Parametri:

m – Matrice (lista di liste di numeri reali)

scalar – Numero reale scalare moltiplicatore

Scopo: Moltiplica una matrice per uno scalare applicando vector-scale a ogni riga. Utilizza mapcar con lambda che applica vector-scale passando ciascuna riga e lo scalare. Operazione di algebra lineare che scala uniformemente tutti gli elementi della matrice. Utilizzata nel Listato 1 dall'algoritmo Levenberg-Marquardt per scalare la matrice identità 3×3 con il parametro di smorzamento λ nella costruzione della matrice modificata H+λI. Il parametro λ varia dinamicamente durante l'ottimizzazione: aumenta se l'errore peggiora (comportamento più cauto tipo GD), diminuisce se l'errore migliora (comportamento più aggressivo tipo NR). Fornisce astrazione per operazione matriciale che altrimenti richiederebbe cicli annidati espliciti.

Valore restituito: Matrice con stesse dimensioni di m, ogni elemento moltiplicato per scalar.

- **matrix-sum (m1 m2)**

Parametri:

m1 – Matrice (lista di liste di numeri reali)

m2 – Matrice con stesse dimensioni di m1

Scopo: Somma due matrici elemento per elemento applicando Somm_2Lis alle coppie di righe corrispondenti. Utilizza mapcar con Somm_2Lis come funzione che prende una riga da m1 e la corrispondente da m2, calcola la somma vettoriale riga per riga. Operazione di algebra lineare fondamentale. Utilizzata nel Listato 1 esclusivamente dall'algoritmo Levenberg-Marquardt per costruire la matrice Hessiana modificata sommando la Hessiana H calcolata nel punto corrente con la matrice identità scalata λI: H_mod = H + λI. Questa modifica regolarizza il sistema lineare da risolvere per calcolare il passo di ottimizzazione, garantendo che la matrice rimanga ben condizionata e invertibile anche quando la Hessiana originale è singolare o quasi-singolare.

Valore restituito: Matrice con stesse dimensioni delle matrici parametro, contenente la somma elemento per elemento.

- **Media_lis_Pu (lis_pu)**

Parametri:

lis_pu – Lista di punti bidimensionali o tridimensionali

Scopo: Calcola il baricentro (centroide) di una lista di punti mediando le coordinate x e y, e utilizzando il valore assoluto per z. Verifica che lis_pu sia lista non vuota, calcola lunghezza n, inizializza accumulatori xm, ym, zma a 0.0. Itera con foreach su tutti i punti: accumula x con car, y con cadr, valore assoluto di z con

abs e caddr. Calcola medie dividendo per n, restituisce punto (xm/n, ym/n, zma/n). L'uso del valore assoluto per z è peculiare e motivato dal contesto: nel Listato 1 è utilizzata per calcolare il punto di partenza delle ottimizzazioni come baricentro dei centri delle tre circonferenze (tutti con z=0 quindi irrilevante); quando applicata a punti con z negativa (metafora orografica) il valore assoluto garantisce che il baricentro abbia quota positiva coerente con interpretazione "vetta media".

Valore restituito: Punto tridimensionale (x_media, y_media, |z|_media) rappresentante il baricentro della lista punti.

- **MS (lst len)**

Parametri:

lst – Lista di numeri reali (tipicamente distanze o errori)

len – Numero intero rappresentante la lunghezza della lista

Scopo: Calcola la Media Semplice (media aritmetica) di una lista di valori. Implementa la formula MS = Σ(xi)/n. Somma tutti gli elementi della lista con apply '+, divide per len. Prima delle quattro metriche statistiche disponibili nel Listato 2. Questa funzione obiettivo è lineare e fornisce una misura diretta della distanza media dai perimetri delle tre circonferenze senza enfatizzare alcun tipo di errore particolare (outlier pesano quanto valori normali). Quando usata per mappare il paesaggio di errore genera superfici con caratteristiche lineari, meno articolate rispetto a metriche quadratiche. Non è sensibile alla distribuzione dell'errore tra le misurazioni, solo alla loro somma. Utilizzata come prima opzione selezionabile (tipo=1) in Test_Pu_LisCir.

Valore restituito: Numero reale rappresentante la media aritmetica (unità di misura = unità coordinate disegno).

N

- **NCorMax_LisPu (LisPu n)**

Parametri:

LisPu – Lista di punti tridimensionali

n – Numero intero indice coordinata (0=x, 1=y, 2=z)

Scopo: Estrae la coordinata massima lungo l'asse n-esimo da una lista di punti. Applica mapcar con lambda che estrae la n-esima coordinata di ciascun punto mediante nth, poi applica max alla lista risultante con apply 'max. Utilizzata in entrambi i listati dalla funzione PuMax_LisPu per costruire il punto con coordinate massime del bounding box applicandola a ciascun asse (0, 1, 2) mediante mapcar. Fornisce astrazione funzionale che evita cicli espliciti per trovare massimi. Se LisPu è vuota o n fuori range, il comportamento è indefinito (errore di apply 'max su lista vuota o accesso nth invalido).

Valore restituito: Numero reale rappresentante il valore massimo della coordinata n-esima tra tutti i punti.

- **NCorMin_LisPu (LisPu n)**

Parametri:

LisPu – Lista di punti tridimensionali

n – Numero intero indice coordinata (0=x, 1=y, 2=z)

Scopo: Estrae la coordinata minima lungo l'asse n-esimo da una lista di punti. Applica mapcar con lambda che estrae la n-esima coordinata di ciascun punto mediante nth, poi applica min alla lista risultante con apply 'min. Utilizzata in entrambi i listati dalla funzione PuMin_LisPu per costruire il punto con coordinate minime del bounding box applicandola a ciascun asse (0, 1, 2) mediante mapcar. Fornisce astrazione funzionale che evita cicli espliciti per trovare minimi. Gemella di NCorMax_LisPu con logica identica ma operatore min. Se LisPu è vuota o n fuori range, il comportamento è indefinito.

Valore restituito: Numero reale rappresentante il valore minimo della coordinata n-esima tra tutti i punti.

O

- **objective-function (point LisCen LisRad)**

Parametri:

point – Punto tridimensionale candidato da valutare

LisCen – Lista di tre punti (centri delle circonferenze di riferimento)

LisRad – Lista di tre numeri reali positivi (raggi delle circonferenze)

Scopo: Calcola la somma dei quadrati degli errori (SSE, Sum of Squared Errors) per un punto candidato rispetto alle tre circonferenze di riferimento. Funzione obiettivo fondamentale che tutti e quattro gli algoritmi di ottimizzazione del Listato 1 mirano a minimizzare. Inizializza accumulatore err a 0.0, itera su indici 0-2 generati da range, per ciascun indice i calcola distanza dal punto al centro i-esimo, sottrae il raggio i-esimo, eleva al quadrato, accumula in err. Implementa SSE = Σ(di-ri)² senza divisione per n né radice quadrata (a differenza di compute-error che calcola EQMR). La forma SSE è equivalente a EQMR per scopi di ottimizzazione (stessa posizione del minimo) ma computazionalmente più semplice evitando divisione e radice. Valori maggiori indicano incoerenza geometrica maggiore.

Valore restituito: Numero reale non negativo rappresentante SSE (unità di misura = unità coordinate al quadrato).

- **optimize-GA (LisCir lay)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

lay – Stringa rappresentante il nome del layer per visualizzazione punto risultante

Scopo: Implementa l'algoritmo Griglia Adattiva con ricerca esaustiva iterativa per il Listato 1. Quarto metodo di ottimizzazione che non utilizza derivate ma campionamento sistematico. Estrae

centri e raggi mediante Get_Data_LisCir, inizializza parametri: interv=32 (semiampiezza griglia iniziale 64×64 con *2), apert=2 (fattore apertura per zoom), cicli=40 (massime iterazioni), tol=1e-6 (tolleranza convergenza spaziale). Calcola rettangolo iniziale mediante Rect_Ini (bounding box punti cardinali tre cerchi), valuta griglia iniziale 64×64 mediante calc_grid_err ottenendo Pu_min (minimo corrente) e Pu_max (punto iniziale per report). Entra in ciclo while che continua finché non converge (verificato da P_tol che controlla vicinanza Pu_min e Pu_max sotto tolleranza) e finché num_cicli < 40. Ad ogni iterazione: genera nuovo rettangolo più piccolo centrato su Pu_min mediante Rect_Gen, valuta griglia 32×32 su rettangolo raffinato, aggiorna Pu_min e Pu_max, incrementa contatore. Disegna punto finale, costruisce risultati.

Valore restituito: Lista associativa contenente risultati ottimizzazione costruita da build-optimization-results.

- **optimize-GD (LisCir start-point lay)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

start-point – Punto tridimensionale iniziale dell'ottimizzazione (baricentro centri)

lay – Stringa rappresentante il nome del layer per visualizzazione punto risultante

Scopo: Implementa l'algoritmo Discesa del Gradiente (Gradient Descent) per il Listato 1. Primo metodo di ottimizzazione, il più semplice e robusto. Registra tempo iniziale, estrae centri e raggi mediante Get_Data_LisCir, inizializza parametri: max-iter=1000, learning-rate=0.01 (passo fisso), tol=1e-6 (tolleranza sul gradiente per convergenza). Inizializza variabili: errors (storia errori, accumulata in ordine inverso), iter-num=0, converged=nil, point=start-point. Entra in ciclo while che continua finché iterazioni < 1000 e non converge. Ad ogni iterazione: calcola gradiente nel punto corrente mediante gradient-function, verifica convergenza controllando se norma gradiente < tol mediante vector-length, se non converge aggiorna punto sottraendo gradiente scalato per learning-rate (movimento verso minimo) mediante Somm_2Lis e vector-scale, accumula errore corrente calcolato da objective-function, incrementa contatore. Calcola durata, disegna punto finale, costruisce risultati.

Valore restituito: Lista associativa contenente risultati ottimizzazione costruita da build-optimization-results.

- **optimize-LM (LisCir start-point lay)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

start-point – Punto tridimensionale iniziale dell'ottimizzazione (baricentro centri)

lay – Stringa rappresentante il nome del layer per visualizzazione punto risultante

Scopo: Implementa l'algoritmo Levenberg-Marquardt per il Listato 1. Terzo metodo di ottimizzazione, il più sofisticato e adattivo. Combina robustezza GD con velocità NR mediante parametro di smorzamento λ che si adatta dinamicamente. Registra tempo, estrae dati cerchi, inizializza: max-iter=1000, tol=1e-6, lambda-init=0.01, lambda-factor=10.0 (fattore moltiplicativo per aggiustamenti λ). Inizializza variabili ottimizzazione e lambda-param=lambda-init. Ciclo while: ad ogni iterazione calcola errore corrente e lo accumula, calcola gradiente e Hessiana, verifica convergenza su norma gradiente. Se non converge: costruisce Hessiana modificata H_mod = H + λI mediante matrix-sum, matrix-scale, matrix-identity; calcola passo approssimato come -grad/(λ+1) (versione semplificata, non risolve sistema completo); calcola nuovo punto e nuovo errore; confronta errori: se nuovo < corrente (miglioramento) diminuisce λ per comportamento più NR e accetta punto, altrimenti aumenta λ per comportamento più GD e rifiuta punto; incrementa iterazioni. Calcola durata, disegna, costruisce risultati includendo lambda-param finale.

Valore restituito: Lista associativa contenente risultati ottimizzazione costruita da build-optimization-results.

- **optimize-NR (LisCir start-point lay)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

start-point – Punto tridimensionale iniziale dell'ottimizzazione (baricentro centri)

lay – Stringa rappresentante il nome del layer per visualizzazione punto risultante

Scopo: Implementa l'algoritmo Newton-Raphson per il Listato 1. Secondo metodo di ottimizzazione, utilizza informazione secondo ordine (curvatura) per convergenza rapida. Registra tempo, estrae dati cerchi, inizializza: max-iter=1000, tol=1e-6. Inizializza variabili ottimizzazione. Ciclo while: ad ogni iterazione calcola e accumula errore corrente, calcola gradiente e Hessiana, verifica convergenza su norma gradiente. Se non converge: scala gradiente per -1.0 (inversione direzione), calcola determinante Hessiana per verificare invertibilità; se determinante quasi nullo (< 1e-10, matrice singolare) usa passo fallback proporzionale a gradiente scalato -0.01, altrimenti risolve sistema lineare H·step = -grad mediante solve-system-3x3 per ottenere passo di Newton ottimale; calcola nuovo punto e nuovo errore; confronta errori: se nuovo < corrente (miglioramento) accetta punto, altrimenti accetta comunque punto dimezzato (compromesso per evitare stallo); incrementa iterazioni. Calcola durata, disegna, costruisce risultati. La gestione del caso singolare e il confronto errori rendono l'algoritmo più robusto rispetto a NR puro.

Valore restituito: Lista associativa contenente risultati ottimizzazione costruita da build-optimization-results.

- **OrdineDimensione (num)**

Parametri:

num – Numero reale o intero di cui determinare l'ordine di grandezza

Scopo: Determina l'ordine di grandezza (potenza di 10 dominante) di un numero mediante iterazione sulla lista di potenze di 10 generata da LisDec_Num. Genera lista decadi chiamando LisDec_Num, calcola valore assoluto del numero, inizializza ordine alla prima potenza (1). Itera con foreach su tutte le potenze di 10 nella lista: per ciascuna potenza d, se d <= num assoluto aggiorna ordine a d. Al termine del ciclo ordine contiene la massima potenza di 10 non superiore a num. Ad esempio, per num=3456 restituisce 1000 (10^3). Utilizzata nel Listato 2 dalla funzione OrdineEQuantita per analizzare l'ordine di grandezza dei valori z del paesaggio di errore (quote delle vette) e determinare automaticamente un passo significativo per le sezioni orizzontali nella visualizzazione delle curve di livello.

Valore restituito: Numero intero potenza di 10 rappresentante l'ordine di grandezza di num (1, 10, 100, 1000, ...).

- **OrdineEQuantita (num)**

Parametri:

num – Numero reale o intero da analizzare

Scopo: Calcola sia l'ordine di grandezza sia il numero di "unità" di quell'ordine contenute nel numero assoluto. Chiama OrdineDimensione per ottenere ordine (potenza di 10), calcola quantità come 1 + parte intera di (|num|/ordine) mediante divisione, abs, fix e somma di 1. La somma di 1 garantisce che quantità sia almeno 1 anche per numeri < ordine. Ad esempio, per num=3456 e ordine=1000, quantità = 1 + fix(3456/1000) = 1 + 3 = 4. Restituisce lista (ordine quantità). Utilizzata nel Listato 2 dalla funzione Sol_Analysis per ottimizzare il numero di sezioni orizzontali da generare: applica OrdineEQuantita ai valori z minimo e massimo del bounding box, calcola range appropriati moltiplicando ordine per quantità e negando (per metafora orografica), determina numero sezioni come 10·quantità per garantire granularità adeguata senza sovraccarico grafico eccessivo.

Valore restituito: Lista di due numeri interi (ordine quantità) dove ordine è potenza di 10 e quantità è il numero di tali unità contenute nel valore assoluto.

P

- **P_ent (x)**

Parametri:

x – Valore di tipo qualsiasi da verificare

Scopo: Verifica se un valore è un'entità AutoCAD valida di tipo ENAME esistente nel database del disegno. Controlla che x sia non-nil e che il suo tipo sia esattamente 'ENAME mediante (= (type x) 'ENAME). Restituisce T se entrambe le condizioni sono soddisfatte, nil altrimenti. Utilizzato pervasivamente in entrambi i listati come guardia per prevenire errori quando si manipolano ENAME: le funzioni Get_DxfVal_Ent, Get_Data_Cir, Get_BouBox_Ent verificano preventivamente il parametro; le funzioni di disegno verificano i risultati di entlast; Ent_Erase verifica prima di eliminare. Fornisce astrazione che centralizza la verifica tipo ENAME evitando errori di runtime quando funzioni AutoCAD native (entget, entdel, vlax-ename->vla-object) ricevono valori non-ENAME. Nota: verifica solo il tipo, non l'esistenza effettiva dell'entità nel database (ENAME potrebbe riferirsi a entità già eliminata).

Valore restituito: T se x è ENAME valido, nil altrimenti.

- **P_Layer (name)**

Parametri:

name – Stringa rappresentante il nome del layer da cercare

Scopo: Verifica l'esistenza di un layer nella tabella dei simboli del disegno AutoCAD. Chiama funzione nativa tblsearch con argomenti "LAYER" (nome tabella simboli) e name (chiave ricerca). Restituisce la definizione completa del layer se esiste (lista associativa contenente tutti gli attributi DXF del layer: nome, colore, tipo linea, flags, ecc.), oppure nil se il layer non esiste nella tabella simboli. Utilizzata in entrambi i listati dalle funzioni di gestione layer Lay_Check e Lay_Make_Col per verificare esistenza prima di decidere se creare nuovo layer o modificare esistente. Fornisce astrazione che incapsula l'accesso alla tabella simboli e la ricerca nominativa. La definizione completa restituita contiene più informazioni del necessario ma può essere utile per ispezioni o modifiche avanzate.

Valore restituito: Lista associativa DXF contenente la definizione del layer se esiste, nil altrimenti.

- **P_Pu (Pu)**

Parametri:

Pu – Valore di tipo qualsiasi da verificare

Scopo: Predicato fondamentale che verifica se un valore è un punto valido (bidimensionale o tridimensionale). Controlla tre condizioni in AND logico: (1) Pu è una lista (listp), (2) la lunghezza è 2 o 3 (member (length Pu) '(2 3)), (3) tutti gli elementi sono numeri (vl-every 'numberp Pu). Restituisce T solo se tutte le condizioni soddisfatte, nil altrimenti. Utilizzato pervasivamente in entrambi i listati come guardia di input per funzioni geometriche: Draw_Point verifica prima di disegnare, Pu_3D verifica prima di normalizzare, le funzioni di manipolazione punti assumono input validi. Fornisce astrazione che centralizza la definizione di "punto valido" in tutto il codice garantendo consistenza. Non distingue tra 2D e 3D, per questo esistono predicati specializzati P_Pu_2D e P_Pu_3D.

Valore restituito: T se Pu è punto valido (lista di 2 o 3 numeri), nil altrimenti.

- **P_Pu_2D (Pu)**

Parametri:

Pu – Valore da verificare (tipicamente punto)

Scopo: Verifica se un valore è un punto bidimensionale valido. Controlla due condizioni in AND: (1) Pu è un punto valido verificato da P_Pu, (2) la lunghezza è esattamente 2 (= (length Pu) 2). Restituisce T solo se entrambe soddisfatte. Specializzazione di P_Pu che garantisce dimensionalità specifica 2D (solo x e y, nessuna coordinata z). Utilizzato in entrambi i listati dalla funzione Pu_3D per decidere se aggiungere z=0.0 al punto (se è 2D) o lasciarlo invariato (se è già 3D). Fornisce astrazione per distinguere punti 2D da 3D senza necessità di controllare manualmente la lunghezza in ogni punto del codice dove la dimensionalità è rilevante.

Valore restituito: T se Pu è punto bidimensionale valido (lista di esattamente 2 numeri), nil altrimenti.

- **P_Pu_3D (Pu)**

Parametri:

Pu – Valore da verificare (tipicamente punto)

Scopo: Verifica se un valore è un punto tridimensionale valido. Controlla due condizioni in AND: (1) Pu è un punto valido verificato da P_Pu, (2) la lunghezza è esattamente 3 (= (length Pu) 3). Restituisce T solo se entrambe soddisfatte. Specializzazione di P_Pu che garantisce dimensionalità specifica 3D (x, y e z). Utilizzato in entrambi i listati dalla funzione Pu_3D per decidere se il punto è già tridimensionale (restituiscilo invariato) o richiede normalizzazione (aggiungi z). Fornisce astrazione simmetrica a P_Pu_2D per classificazione completa degli stati dimensionali possibili di un punto. Insieme, i tre predicati P_Pu, P_Pu_2D, P_Pu_3D formano una gerarchia di validazione progressivamente più specifica.

Valore restituito: T se Pu è punto tridimensionale valido (lista di esattamente 3 numeri), nil altrimenti.

- **P_tol (Pu_min Pu_max tol)**

Parametri:

Pu_min – Primo punto tridimensionale da confrontare

Pu_max – Secondo punto tridimensionale da confrontare

tol – Numero reale positivo rappresentante tolleranza (soglia di vicinanza)

Scopo: Verifica se due punti tridimensionali sono sufficientemente vicini rispetto a una tolleranza specificata, controllando sia le differenze coordinate per coordinata sia la distanza euclidea complessiva. Esegue quattro controlli in AND logico: (1) differenza assoluta x < tol, (2) differenza assoluta y < tol, (3) differenza assoluta z < tol, (4) distanza euclidea 3D < tol. Restituisce T solo se tutti e quattro i controlli soddisfatti, garantendo vicinanza rigorosa. Utilizzata esclusivamente nel Listato 1 dall'algoritmo Griglia Adattiva optimize-GA come criterio di convergenza: verifica se il punto di minimo corrente Pu_min è sufficientemente vicino al punto di riferimento Pu_max (che viene aggiornato ad ogni raffinamento) per interrompere l'iterazione. La combinazione di controlli per-coordinata e distanza globale garantisce che i punti siano vicini in senso stretto (non basta vicinanza su un solo asse).

Valore restituito: T se i due punti sono più vicini di tol su tutte le coordinate e in distanza euclidea, nil altrimenti.

- **princ-advanced-results (all-results)**

Parametri:

all-results – Lista di quattro liste associative contenenti i risultati delle ottimizzazioni GD, LM, NR, GA

Scopo: Funzione di coordinamento di alto livello che orchestra la stampa dell'intero report comparativo del Listato 1 in undici sezioni strutturate. Invoca nell'ordine: intestazione principale con doppio ===== (24 unità = 120 caratteri) e titolo centrato "COMPARAZIONE DEI RISULTATI DELLE TRE OTTIMIZZAZIONI", separatore =====, intestazione colonne metodo/punto-partenza/errore/punto-calcolato/errore, separatore -----, righe dati per ogni metodo mediante princ-results-row, separatore -----, analisi convergenza mediante princ-methods-with-convergence, separatore -----, identificazione metodo migliore mediante princ-best-method, separatore -----, intestazione raggi, separatore -----, tabella raggi mediante princ-radius, separatore -----, intestazione residui, separatore -----, tabella residui mediante princ-residuals-by-method, chiusura =====. Utilizza i separatori grafici per delimitare visivamente le sezioni creando struttura gerarchica leggibile. Tutte le funzioni princ-* stampano direttamente su console, questa funzione coordina solo la sequenza e i separatori.

Valore restituito: Nessuno (funzione di side-effect che stampa report completo su console).

- **princ-best-method (all-results)**

Parametri:

all-results – Lista di quattro liste associative contenenti i risultati delle ottimizzazioni

Scopo: Identifica e annuncia il metodo con l'errore finale minore nel Listato 1 mediante iterazione su tutti i risultati e confronto numerico. Inizializza min-error a valore molto grande (1e99, praticamente infinito) e best-name a nil. Itera con foreach su tutti i risultati: per ciascuno estrae error e name_method dalla lista associativa mediante Get_LisVal_LisAssoc, confronta error con min-error corrente, se minore aggiorna sia min-error sia best-name. Al termine del ciclo best-name contiene il nome del metodo con errore minore. Stampa messaggio formattato "Il metodo con l'errore minore è: [NOME] con errore [VALORE]" utilizzando rtos-pad con precisione 10 decimali e larghezza 6 per allineamento. Fornisce indicazione chiara e immediata della soluzione migliore tra le quattro calcolate, utile per confronto rapido senza analisi dettagliata della tabella completa.

Valore restituito: Nessuno (funzione di side-effect che stampa identificazione metodo migliore).

- **princ-methods-with-convergence (all-results)**

Parametri:

all-results – Lista di quattro liste associative contenenti i risultati delle ottimizzazioni

Scopo: Stampa la sezione di analisi della convergenza del Listato 1 mostrando per ogni metodo: stato di convergenza (SI/NO), durata calcolo formattata in minuti e secondi, numero di iterazioni con padding. Itera con foreach su tutti i risultati (nota: parametro chiamato result-pair ma è singolo result, naming inconsistente). Per ciascun result estrae name_method, iterations, converged, duration mediante Get_LisVal_LisAssoc dalla porzione (cdr result-pair) (accesso peculiare, forse bug o struttura dati annidata non documentata). Calcola minuti interi e secondi resto dividendo duration per 60 con fix e rem. Stampa riga formattata: nome metodo, "convergenza raggiunta: SI/NO", "durata calcolo: X minuti e Y.ZZ secondi" (omette minuti se zero), "numero di iterazioni: NNNN" (padding 4 cifre con itoa-pad). Fornisce analisi prestazionale e di affidabilità: convergenza indica se algoritmo ha raggiunto soluzione stabile, durata e iterazioni indicano efficienza computazionale.

Valore restituito: Nessuno (funzione di side-effect che stampa analisi convergenza).

- **princ-radius (all-results)**

Parametri:

all-results – Lista di quattro liste associative contenenti i risultati delle ottimizzazioni

Scopo: Stampa i raggi delle tre circonferenze di riferimento e il raggio medio nel Listato 1 per fornire contesto geometrico. Estrae radius_list dalla prima lista associativa (tutti i risultati condividono stessi raggi quindi basta estrarre da uno qualsiasi) mediante Get_LisVal_LisAssoc. Calcola raggio medio come somma raggi diviso per 3 con apply '+ e divisione. Inizializza stringa vuota, itera con foreach su tutti i raggi concatenando ciascun raggio formattato con rtos modalità 2 precisione 6 decimali seguito da 6 spazi per separazione. Stampa riga con padding iniziale di 28 spazi per allineamento con intestazione, stringa raggi, separatore " - ", raggio medio formattato. Fornisce informazione utile per interpretare gli errori: errori piccoli rispetto ai raggi indicano buona accuratezza, errori comparabili ai raggi indicano problema mal condizionato.

Valore restituito: Nessuno (funzione di side-effect che stampa tabella raggi).

- **princ-residuals-by-method (all-results)**

Parametri:

all-results – Lista di quattro liste associative contenenti i risultati delle ottimizzazioni

Scopo: Stampa la sezione di analisi dettagliata degli errori del Listato 1 mostrando per ogni metodo: i tre residui individuali (uno per circonferenza) e la media dei valori assoluti. Itera con foreach su tutti i risultati, per ciascuno estrae name_method e residuals mediante Get_LisVal_LisAssoc. Calcola media valori assoluti applicando abs a ciascun residuo con mapcar, sommando con apply '+, dividendo per 3. Stampa riga formattata: nome metodo, tre residui con rtos-pad modalità 2 precisione 8 decimali larghezza 4 separati da 3 spazi, separatore " - ", media valori assoluti con stessa formattazione. I residui possono essere positivi (punto esterno alla circonferenza), negativi (punto interno), o quasi nulli (punto sulla circonferenza). Questa analisi evidenzia come l'errore si distribuisce tra le tre misurazioni permettendo di identificare circonferenze problematiche: se un residuo è molto maggiore degli altri indica possibile errore in quella misurazione specifica.

Valore restituito: Nessuno (funzione di side-effect che stampa tabella residui dettagliata).

- **princ-results-row (all-results)**

Parametri:

all-results – Lista di quattro liste associative contenenti i risultati delle ottimizzazioni

Scopo: Stampa la prima sezione della tabella comparativa del Listato 1 mostrando per ogni metodo: nome, punto di partenza, errore iniziale, punto finale, errore finale. Itera con foreach su tutti i risultati, per ciascuno estrae name_method, start_point, start_error, point, error mediante Get_LisVal_LisAssoc. Stampa riga formattata concatenando: "\n", nome metodo (già con padding a 20 caratteri nelle costanti stringa), " ", punto di partenza formattato 2D con format-point (nil come secondo parametro esclude z), " ", errore iniziale con rtos-pad modalità 2 precisione 4 decimali larghezza 6, " ", punto finale formattato 2D, " ", errore finale con rtos-pad modalità 2 precisione 16 decimali larghezza 6, " ". La precisione alta (16 decimali) per errore finale permette di distinguere differenze minime tra metodi. Formattazione con padding garantisce allineamento colonnare perfetto anche con valori molto diversi.

Valore restituito: Nessuno (funzione di side-effect che stampa righe tabella risultati).

- **Pu_2D (Pu)**

Parametri:

Pu – Punto bidimensionale o tridimensionale

Scopo: Converte un punto in formato bidimensionale forzando z=0.0 e mantenendo x e y. Estrae prima coordinata con car, seconda con cadr, costruisce lista di tre elementi aggiungendo 0.0 come terzo. Nota: nonostante il nome "2D" la funzione restituisce punto tridimensionale con z forzata a zero, non punto bidimensionale puro (che avrebbe solo due elementi). Utilizzata esclusivamente nel Listato 1 dalle funzioni di calcolo errore compute-error e compute-residuals per garantire che le distanze siano calcolate solo sul piano xy ignorando la coordinata z dei punti candidati (che potrebbe essere non-zero per motivi interni agli algoritmi). Fornisce astrazione per proiezione sul piano xy. Funzione gemella concettuale di Pu_3D ma con semantica diversa (forza z=0 invece di garantire esistenza z).

Valore restituito: Punto tridimensionale (x, y, 0.0) dove x e y sono le coordinate originali del parametro.

- **Pu_3D (Pu)**

Parametri:

Pu – Punto bidimensionale o tridimensionale

Scopo: Garantisce la tridimensionalità di un punto mediante normalizzazione: se il punto è bidimensionale (verificato da P_Pu_2D) aggiunge coordinata z=0.0 mediante append, se è già tridimensionale (verificato da P_Pu_3D) lo restituisce invariato. Utilizza cond per discriminazione mutualmente esclusiva. La tridimensionalità è richiesta da AutoCAD per coerenza anche quando le operazioni sono planari, e da molte funzioni matematiche che assumono punti 3D. Utilizzata pervasivamente in entrambi i listati: dalle funzioni di costruzione codici DXF (cons_10, cons_11) per garantire formato corretto prima di creare entità, dalle funzioni geometriche che richiedono tre coordinate, dalle funzioni di conversione che producono safearray ActiveX richiedenti tridimensionalità. Fornisce astrazione che centralizza la logica di normalizzazione 2D→3D evitando controlli e conversioni sparse nel codice.

Valore restituito: Punto tridimensionale (x, y, z) dove z=0.0 se punto originale era 2D, z originale se era già 3D.

- **PuMax_LisPu (LisPu)**

Parametri:

LisPu – Lista di punti tridimensionali

Scopo: Calcola il punto con le coordinate massime di una lista di punti restituendo il vertice massimo del bounding box. Applica mapcar con lambda che per ciascun asse n in (0 1 2) chiama NCorMax_LisPu passando LisPu e n, ottenendo la coordinata massima su quell'asse. Assembla le tre coordinate massime in un nuovo punto (x_max, y_max, z_max). Questo punto non necessariamente appartiene alla lista originale (è la "fusione" dei massimi indipendenti per ciascun asse). Utilizzata in entrambi i listati dalla funzione LisPuBox_LisPu per costruire il secondo punto del bounding box (vertice opposto a quello minimo calcolato da PuMin_LisPu). Fornisce astrazione funzionale simmetrica a PuMin_LisPu per calcolo efficiente degli estremi senza cicli espliciti.

Valore restituito: Punto tridimensionale (x_max, y_max, z_max) con coordinate massime su ciascun asse indipendentemente.

- **PuMin_LisPu (LisPu)**

Parametri:

LisPu – Lista di punti tridimensionali

Scopo: Calcola il punto con le coordinate minime di una lista di punti restituendo il vertice minimo del bounding box. Applica mapcar con lambda che per ciascun asse n in (0 1 2) chiama NCorMin_LisPu passando LisPu e n, ottenendo la coordinata minima su quell'asse. Assembla le tre coordinate minime in un nuovo punto (x_min, y_min, z_min). Questo punto non necessariamente appartiene alla lista originale (è la "fusione" dei minimi indipendenti per ciascun asse). Utilizzata in entrambi i listati dalla funzione LisPuBox_LisPu per costruire il primo punto del bounding box (vertice opposto a quello massimo calcolato da PuMax_LisPu). Fornisce astrazione funzionale per calcolo efficiente degli estremi senza cicli espliciti.

Valore restituito: Punto tridimensionale (x_min, y_min, z_min) con coordinate minime su ciascun asse indipendentemente.

R

- **range (a b)**

Parametri:

a – Numero intero iniziale (incluso)

b – Numero intero finale (incluso)

Scopo: Genera una lista di numeri interi consecutivi da a a b inclusi mediante iterazione e accumulo. Inizializza lista accumulatore out a nil, entra in ciclo while che continua finché a <= b: accumula a corrente in out mediante cons (inserisce in testa), incrementa a di 1 con 1+. Esce dal ciclo quando a > b. Inverte lista con reverse per ottenere ordine crescente (senza inversione sarebbe decrescente per accumulo in testa). Ad esempio, (range 0 2) genera (0 1 2). Funzione di utilità per iterazioni e generazione indici utilizzata pervasivamente in entrambi i listati: per iterare su indici 0-2 delle tre circonferenze negli algoritmi, per generare indici righe/colonne nella griglia di errore, per costruire matrici identità, per applicare funzioni su sequenze di coordinate. Fornisce astrazione funzionale che evita cicli manuali con contatori.

Valore restituito: Lista di numeri interi consecutivi [a, a+1, a+2, ..., b] se a <= b, lista vuota se a > b.

- **Rect_Gen (Pu-min rect-min rect-max multipl divisions)**

Parametri:

Pu-min – Punto bidimensionale (x y) del minimo corrente trovato nella griglia

rect-min – Punto bidimensionale (x y) vertice minimo del rettangolo precedente

rect-max – Punto bidimensionale (x y) vertice massimo del rettangolo precedente

multipl – Numero reale moltiplicatore per fattore di apertura (tipicamente 2)

divisions – Numero intero divisioni griglia (tipicamente 32)

Scopo: Genera un nuovo rettangolo di ricerca centrato sul punto di minimo corrente con dimensioni ridotte secondo un fattore di raffinamento per l'algoritmo GA del Listato 1. Calcola le dimensioni del rettangolo precedente (larghezza = rect-max.x - rect-min.x, altezza = rect-max.y - rect-min.y), calcola il passo di raffinamento come (dimensione / divisions) * multipl. Costruisce nuovo rettangolo con vertice minimo = Pu-min - (passo, passo) e vertice massimo = Pu-min + (passo, passo). Questo "zooma" sulla regione promettente: se multipl=2 e divisions=32, il nuovo rettangolo ha dimensioni 2/32 = 1/16 del precedente. Utilizzata iterativamente da optimize-GA per implementare il meccanismo di raffinamento progressivo: ad ogni ciclo centra un rettangolo più

piccolo sul minimo corrente permettendo campionamento più fine della zona critica convergendo verso la soluzione con precisione crescente.

Valore restituito: Lista di due punti bidimensionali (nuovo-rect-min nuovo-rect-max) definenti il rettangolo raffinato.

- **Rect_Ini (LisCir)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE

Scopo: Calcola il rettangolo iniziale di ricerca per l'algoritmo GA del Listato 1 come bounding box dell'unione dei punti cardinali delle tre circonferenze. Applica mapcar 'LisPuQuad_Cir alla lista cerchi ottenendo lista di tre liste di quattro punti (12 punti totali), appiattisce la struttura con apply 'append ottenendo lista piatta di 12 punti, applica LisPuBox_LisPu che calcola il bounding box (punto minimo e massimo) di tutti i 12 punti. Questo approccio garantisce che il rettangolo iniziale racchiuda completamente tutte e tre le circonferenze campionando i loro punti estremi (nord, sud, est, ovest). Definisce la regione candidata iniziale massimale per la ricerca: qualunque soluzione geometricamente sensata della trilaterazione deve cadere in questa regione o nelle sue immediate vicinanze. Utilizzata una sola volta all'inizio di optimize-GA per inizializzare la ricerca.

Valore restituito: Lista di due punti bidimensionali (rect-min rect-max) definenti il rettangolo di contenimento iniziale.

- **rtos-pad (num mode precision total-width)**

Parametri:

num – Numero reale da convertire in stringa

mode – Numero intero modalità conversione (2=decimale, 1=scientifico, 3=ingegneria, 4=architettura)

precision – Numero intero cifre decimali desiderate

total-width – Numero intero larghezza totale desiderata della parte intera (per padding spazi)

Scopo: Converte un numero reale in stringa con modalità, precisione decimale e larghezza totale specificate, aggiungendo padding di spazi a sinistra per parte intera e zeri a destra per parte decimale. Funzione complessa di formattazione utilizzata pervasivamente nel Listato 1 per allineare valori numerici nelle tabelle comparative. Chiama rtos nativa per conversione base. Se mode=2 (decimale) e precision>0: localizza punto decimale con vl-string-position, se assente aggiunge punto, calcola cifre decimali presenti, completa con zeri mediante make-zeros fino a precision. Localizza nuovamente punto decimale (o usa lunghezza totale se assente), estrae parte intera, calcola padding necessario come total-width - lunghezza-parte-intera, genera spazi con make-spaces, concatena padding + numero completo. Gestisce casi edge: numeri senza decimali, precision=0, total-width insufficiente.

Valore restituito: Stringa contenente spazi di padding seguiti dal numero formattato con precisione decimale garantita, lunghezza parte intera >= total-width.

S

- **select-three-circles ( )**

Parametri: Nessuno (funzione interattiva)

Scopo: Gestisce la selezione interattiva guidata delle tre circonferenze di riferimento per il Listato 1 con validazione tipo e gestione annullamento. Versione specializzata fissa per tre circonferenze (a differenza di Get_LisCir che è parametrica). Inizializza variabili LisCir (accumulatore), ent (entità corrente), done (flag terminazione). Entra in ciclo while che continua finché non terminato e lista non completa (< 3 elementi). Per ogni iterazione richiede selezione mediante entsel con prompt dinamico "Seleziona cerchio N di 3", estrae ENAME con car. Gestisce tre casi mediante cond: (1) se ent è nil (annullamento), verifica se lista parzialmente popolata (mostra messaggio, azzera lista, termina) o termina silenziosamente; (2) se ent non nil, verifica tipo mediante Get_DxfVal_Ent codice 0, se "CIRCLE" accumula e stampa feedback con vl-princ-to-string dell'ENAME, altrimenti stampa errore. Esce restituendo lista completa o nil se annullato.

Valore restituito: Lista di tre ENAME di entità CIRCLE se selezione completata, nil se annullata.

- **Sol_Analysis (LisCir dens fuob DrawSez lay1 lay2 lay3 lay4)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

dens – Numero intero esponente per densità griglia (4-8 → $2^n \times 2^n$ punti)

fuob – Codice numerico funzione obiettivo (1=MS, 2=EQMR, 3=SQM, 4=ICES)

DrawSez – Booleano (T per generare sezioni orizzontali, nil per omettere Fase 5)

lay1 – Stringa layer per superficie loftata

lay2 – Stringa layer per solido volumetrico

lay3 – Stringa layer per sezioni complessive (passo maggiore)

lay4 – Stringa layer per sezioni apicali (passo minore, dettaglio)

Scopo: Funzione di coordinamento di alto livello che orchestra l'intera pipeline volumetrica del Listato 2 in cinque fasi. Gestisce backup e ripristino variabili sistema osmode (16384) e 3dosmode (1). Applica limitatore di sicurezza vincolando dens tra 3 e 9 con min e max per prevenire blocchi (griglia massima 512×512). Fase 1: genera griglia 3D di punti errore mediante Build_Grid passando densità 2^dens. Calcola bounding box griglia con LisPuBox_LisPu. Fase 2: costruisce rete spline mediante Draw_Spl_From_Grid con grado 3 (cubica) su layer "0" temporaneo. Fase 3: crea superficie loftata mediante Draw_SurNet_From_Spl con flag T (elimina spline) su lay1. Fase 4: costruisce solido mediante Draw_Sol_From_Surnet su lay2. Se DrawSez è T, Fase 5: calcola ordine-quantità per z minima e massima con OrdineEQuantita, costruisce range negativi, calcola numero sezioni come

10×quantità, genera sezioni complessive e apicali su lay3 e lay4 mediante due chiamate a Draw_Sections_From_Solid. Ripristina variabili sistema.

Valore restituito: Nessuno (funzione di side-effect che genera geometria complessa nel disegno).

- **solve-system-3x3 (A b) (COMPLETAMENTO)**

Parametri:

A – Matrice 3×3 (lista di tre liste di tre numeri reali) dei coefficienti del sistema lineare

b – Vettore 3D (lista di tre numeri reali) dei termini noti

Scopo: Risolve il sistema lineare Ax=b con matrice 3×3 mediante regola di Cramer calcolando quattro determinanti. Estrae i nove elementi della matrice e i tre elementi del vettore mediante mapcar 'set con apply 'append, associandoli a variabili simboliche (a11-a33, b1-b3). Calcola determinante della matrice coefficienti det-A mediante determinant-3x3. Gestisce caso singolare: se |det-A| < 1e-10 (matrice quasi-singolare, sistema indeterminato o impossibile) restituisce passo di fallback come vettore b scalato per 0.01 mediante vector-scale (movimento cauto proporzionale al gradiente). Se det-A sufficiente, applica regola di Cramer: x[i] = det(A_i)/det(A) dove A_i è matrice con colonna i sostituita da b. Calcola tre determinanti sostituendo colonne 1, 2, 3 con b, divide ciascuno per det-A, assembla soluzione (x1 x2 x3). Utilizzata nel Listato 1 dall'algoritmo Newton-Raphson per calcolare il passo di ottimizzazione risolvendo H·step = -grad dove H è Hessiana e -grad è gradiente negato. La gestione singolarità previene crash quando Hessiana degenera.

Valore restituito: Vettore tridimensionale soluzione del sistema Ax=b se det(A) sufficiente, vettore b·0.01 come fallback se matrice singolare.

- **Somm_2Lis (Lis1 Lis2)**

Parametri:

Lis1 – Lista di numeri (tipicamente coordinate di un punto o componenti di un vettore)

Lis2 – Lista di numeri della stessa lunghezza di Lis1

Scopo: Calcola la somma elemento per elemento di due liste (operazione vettoriale addizione) applicando l'operatore + mediante mapcar. Utilizzata nel Listato 1 per aggiornamenti di posizione negli algoritmi di ottimizzazione: GD aggiunge gradiente scalato negativo al punto corrente, NR e LM aggiungono passo calcolato, gradient-function accumula contributi vettoriali parziali. Fornisce astrazione per operazione vettoriale fondamentale senza necessità di cicli espliciti, sfruttando la natura funzionale di AutoLISP. Le liste devono avere la stessa lunghezza altrimenti mapcar si arresta alla lista più corta. Funzione gemella simmetrica di Diff_2Lis (sottrazione). Utilizzata anche da matrix-sum per sommare matrici riga per riga.

Valore restituito: Lista di numeri della stessa lunghezza delle liste parametro, contenente le somme elemento per elemento (Lis1[i] + Lis2[i]).

- **SQM (lst len)**

Parametri:

lst – Lista di numeri reali (tipicamente distanze o errori)

len – Numero intero rappresentante la lunghezza della lista

Scopo: Calcola lo Scarto Quadratico Medio (deviazione standard) di una lista di valori. Implementa la formula SQM = sqrt(Σ(xi-μ)²/n) dove μ è la media. Calcola media μ sommando elementi e dividendo per len, calcola scarti elevando al quadrato le differenze (x-μ) mediante mapcar con lambda, somma i quadrati con apply '+, divide per len, estrae radice quadrata. Terza delle quattro metriche statistiche disponibili nel Listato 2. Questa funzione obiettivo misura la dispersione (variabilità) delle distanze rispetto alla loro media, ignorando completamente la magnitudo dell'errore assoluto. Un punto equidistante dalle tre circonferenze ha SQM=0 (dispersione nulla) indipendentemente dalla distanza. Quando usata per mappare il paesaggio di errore evidenzia zone di equilibrio geometrico dove le distanze sono omogenee. La struttura intrinseca trova naturalmente punti sulla bisettrice curvilinea (luogo dei punti equidistanti), collegandosi al problema di Apollonio. Utilizzata sia direttamente (tipo=3) sia come componente di ICES.

Valore restituito: Numero reale non negativo rappresentante lo SQM/deviazione standard (unità di misura = unità coordinate disegno).

T

- **Test_Pu_LisCir (LisCir Pu1 tipo)**

Parametri:

LisCir – Lista di tre entità AutoCAD di tipo CIRCLE (riferimenti per trilaterazione)

Pu1 – Punto bidimensionale o tridimensionale candidato da valutare

tipo – Codice numerico funzione obiettivo (1=MS, 2=EQMR, 3=SQM, 4=ICES)

Scopo: Valuta la funzione obiettivo scelta per un punto candidato rispetto alle tre circonferenze di riferimento nel Listato 2. Funzione chiave per la generazione della griglia di errore nella Fase 1 della pipeline volumetrica. Calcola le tre distanze dai perimetri delle circonferenze mediante mapcar applicando lambda che per ciascun cerchio chiama metodo ActiveX vlax-curve-getClosestPointTo passando il cerchio e il punto, poi calcola distance tra punto e punto più vicino sulla curva (questa distanza è lo scostamento dal perimetro, equivalente a |distanza-dal-centro - raggio|). Calcola lunghezza lista distanze. Applica mediante cond la metrica statistica appropriata secondo tipo: 1→MS, 2→EQMR, 3→SQM, 4→ICES, passando lista distanze e lunghezza. Utilizzata massicciamente da Build_Grid che la chiama per ogni punto della griglia (migliaia o decine di migliaia di volte), quindi l'efficienza è critica.

Valore restituito: Numero reale rappresentante il valore della funzione obiettivo selezionata per il punto (unità di misura dipende dalla metrica).

V

- **vector-length (v)**

Parametri:

v – Vettore (lista di numeri reali)

Scopo: Calcola la lunghezza (norma euclidea) di un vettore. Implementa la formula $||v|| = sqrt(\Sigma(xi^2))$. Eleva al quadrato ciascun elemento del vettore mediante mapcar con lambda (lambda (x) (* x x)), somma i quadrati con apply '+, estrae radice quadrata con sqrt. Utilizzata nel Listato 1 per verificare la convergenza degli algoritmi di ottimizzazione GD, NR, LM controllando se la norma del gradiente è sotto la soglia di tolleranza (1e-6): gradiente con norma piccola indica che si è vicini a un punto stazionario (minimo, massimo o sella). Utilizzata anche internamente da gradient-function e hessian-function per calcolare la distanza del punto dai centri delle circonferenze. Fornisce astrazione per calcolo norma vettoriale standard senza cicli espliciti, generalizzabile a vettori di qualsiasi dimensione.

Valore restituito: Numero reale non negativo rappresentante la norma euclidea $||v|| = sqrt(v_1^2 + v_2^2 + ... + v_n^2)$.

- **vector-scale (v scalar)**

Parametri:

v – Vettore (lista di numeri reali)

scalar – Numero reale scalare moltiplicatore

Scopo: Moltiplica un vettore per uno scalare, restituendo un nuovo vettore con tutti gli elementi scalati. Applica mapcar con lambda (lambda (x) (* x scalar)) che moltiplica ciascun elemento per scalar. Operazione di algebra lineare fondamentale utilizzata pervasivamente nel Listato 1. Utilizzata da GD per scalare il gradiente con il learning rate (controllo ampiezza passo), da NR e LM per negare il gradiente (scalar=-1.0) o scalare passi calcolati, da gradient-function per scalare contributi vettoriali parziali, da matrix-scale per scalare righe di matrici. Utilizzata anche da solve-system-3x3 per generare passo di fallback quando matrice singolare (scalar=0.01). Fornisce astrazione per operazione vettoriale che altrimenti richiederebbe cicli espliciti, generalizzabile a vettori di qualsiasi dimensione.

Valore restituito: Vettore con stessa lunghezza di v, ogni elemento moltiplicato per scalar: $(v_1 \cdot scalar, v_2 \cdot scalar, ..., v_n \cdot scalar)$.

CODICE DEL LISTATO 1

```
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(vl-load-com)

(setvar "cmdecho" 0)

(setvar "pdmode" 34)

(setvar "pdsize" -2.5)

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Media_lis_Pu (lis_pu / xm ym zma n)

  "Calcola il baricentro di una lista di punti. Utilizza il valore assoluto per Z"

  (setq n (length lis_pu))

  (if (and (listp lis_pu) (> n 0))

     (progn  (setq xm 0.0 ym 0.0 zma 0.0)

          (foreach pu lis_pu   (setq xm (+ xm (car pu)))            ; somma delle x

                               (setq ym (+ ym (cadr pu)))           ; somma delle y

                               (setq zma (+ zma (abs (caddr pu))))  ; somma dei valori assoluti delle z

          (list (/ xm n) (/ ym n) (/ zma n)))))


;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Somm_2Lis      (Lis1 Lis2) (mapcar '+          Lis1 Lis2))

(defun Diff_2Lis      (Lis1 Lis2) (mapcar '-          Lis1 Lis2))

(defun P_Layer (name) (tblsearch "LAYER" name))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Lay_Make_Col     (lay col)  (if (not (P_Layer lay))  (command "._-layer" "_m" lay "_c" (if (and col (> col 0)) col 7) lay "")

                                        (command "._-layer"          "_c" (if (and col (> col 0)) col 7) lay ""))                lay)

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Lay_Check        (lay) (if (not (P_Layer lay)) (Lay_Make_Col lay 7)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun cons_8   (lay) (cons 8 lay))

(defun cons_10  (Pu1) (cons 10 (Pu_3D Pu1)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun cons_0   (cod / ename)  (if (setq ename (cdr (assoc cod '( (1 . "POLYLINE") (2 . "VERTEX") (3 . "SEQEND") (4 . "LWPOLYLINE")

                                                                 (5 . "LINE") (6 . "CIRCLE") (7 . "POINT") (8 . "SPLINE") (9 .

"ELLIPSE")))))                (cons 0 ename)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun cons_100 (cod / cname)  (if (setq cname (cdr (assoc cod '( (1 . "AcDbEntity") (2 . "AcDb3dPolyline") (3 . "AcDbVertex") (4 .

"AcDb3dPolylineVertex")

                                                                 (5 . "AcDbPolyline") (6 . "AcDbSpline") (7 . "AcDbEllipse")))))

(cons 100 cname)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
(defun Get_LisVal_LisAssoc    (LisAssoc Liskey)  (mapcar '(lambda (key) (cdr (assoc key LisAssoc))) Liskey))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun P_ent (x) (and x (= (type x) 'ENAME)))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Get_DxfVal_Ent           (ent code)  (if (P_ent ent)    (car (Get_LisVal_LisAssoc (entget ent) (list code)))  ))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Get_Data_Cir    (ent)       (if (P_ent ent) (Get_LisVal_LisAssoc (entget ent) (list 10 40))))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Get_Data_LisCir (LisCir)     (apply 'mapcar (cons 'list (mapcar 'Get_Data_Cir LisCir))))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Draw_Point (Pu1 lay)          (if (P_Pu Pu1) (progn (Lay_Check lay) (entmakex (list (cons_0 7) (cons_100 1) (cons_10 Pu1) (cons_8
lay))))))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun format-point (Pu1 3D)  (strcat "(" (rtos-pad (car Pu1) 2 6 4) " , " (rtos-pad (cadr Pu1) 2 6 4) (if 3D (strcat " , " (rtos-pad (caddr Pu1) 2 8
4)) "")  ")"))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun compute-error     (Pu1 LisCen LisRad)  (sqrt (/ (apply '+ (mapcar '(lambda (cen rad) (expt (- (distance (Pu_2D Pu1) cen) rad) 2)) LisCen
LisRad))  (length LisCen))))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun compute-residuals (Pu1 LisCen LisRad)  (mapcar '(lambda (cen rad) (- (distance (Pu_2D Pu1) cen) rad)) LisCen LisRad))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun make-spaces (i / str) (setq str "") (repeat i (setq str (strcat str " "))) str)

(defun make-zeros  (i / str) (setq str "") (repeat i (setq str (strcat str "0"))) str)

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun itoa-pad (num width)  (strcat (make-spaces (- width (strlen (itoa num)))) (itoa num)))

;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun rtos-pad (num mode precision total-width / num-str decimal-pos int-part)

  (setq num-str (rtos num mode precision))


  (if (and (= mode 2) (> precision 0))

     (progn  (setq decimal-pos (vl-string-position (ascii ".") num-str))

          (if (= decimal-pos nil)

             (progn  (setq num-str     (strcat num-str "."))

                  (setq decimal-pos (strlen num-str))))

          (setq num-str (strcat num-str (make-zeros (- precision (- (strlen num-str) decimal-pos 1)))))))


  (if (setq decimal-pos (vl-string-position (ascii ".") num-str))

     (setq int-part (substr num-str 1 decimal-pos))

     (setq int-part num-str))
```

```
  (strcat (make-spaces (- total-width (strlen int-part))) num-str))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun range (a b / out)  (setq out nil)  (while (<= a b) (setq out (cons a out) a (1+ a)))  (reverse out))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_Pu      (Pu)     (and (listp Pu) (member (length Pu) '(2 3)) (vl-every 'numberp Pu)))
(defun P_Pu_2D   (Pu)     (and (P_Pu Pu) (= (length Pu) 2)))
(defun P_Pu_3D   (Pu)     (and (P_Pu Pu) (= (length Pu) 3)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_3D     (Pu)     (cond ((P_Pu_2D Pu) (append Pu '(0.0))) ((P_Pu_3D Pu) Pu)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Pu_2D     (Pu)     (list (car Pu) (cadr Pu) 0.0))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun vector-scale  (v scalar)     (mapcar (function (lambda (x) (* x scalar))) v))  ;Moltiplica un vettore per uno scalare
(defun vector-length (v)            (sqrt (apply '+ (mapcar (function (lambda (x) (* x x))) v))))  ;Calcola la lunghezza (norma) di un vettore
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun matrix-identity (dim)        (mapcar (function (lambda (i) (mapcar (function (lambda (j) (if (= i j) 1.0 0.0))) (range 0 (1- dim))))) (range 0 (1- dim))))
(defun matrix-scale    (m scalar)   (mapcar (function (lambda (row) (vector-scale row scalar))) m))
(defun matrix-sum      (m1 m2)      (mapcar 'Somm_2Lis m1 m2))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun objective-function    (point LisCen LisRad / err i)     (setq err 0.0) (foreach i (range 0 2) (setq err  (+ err (expt (- (distance point (nth i LisCen)) (nth i LisRad)) 2)))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun gradient-function (point LisCen LisRad / grad i diff dist term grad-term)
  (setq grad (list 0.0 0.0 0.0))
  (foreach i (range 0 2)
    (setq diff (Diff_2Lis point (nth i LisCen))
          dist (vector-length diff))
    (if (> dist 1e-10)
       (setq term      (* 2.0 (- dist (nth i LisRad)))
             grad-term (vector-scale diff (/ term dist))
             grad      (Somm_2Lis grad grad-term))))
  grad)
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun hessian-function (point LisCen LisRad / H i Cen Rad diff dist term1 term2 j k H-row H-row-new ValCur diff-j diff-k ValNew)
  (setq H (list (list 0.0 0.0 0.0) (list 0.0 0.0 0.0) (list 0.0 0.0 0.0)))
  (foreach i (range 0 2)
    (setq center (nth i LisCen)
```

```
        radius (nth i LisRad)
        diff (Diff_2Lis point center)
        dist (vector-length diff))
   (if (> dist 1e-10) ; Evita divisione per zero
      (progn
       (setq term1 (/ (* 2.0 (- dist radius)) dist)
            term2 (/ (* 2.0 radius) (expt dist 3)))
       (foreach j (range 0 2)
         (setq H-row (nth j H))
         (setq H-row-new H-row)
         (foreach k (range 0 2)
           (setq current-val (nth k H-row)
                 diff-j      (nth j diff)
                 diff-k      (nth k diff))
           (if (= j k)
              (setq new-val (+ current-val term1 (* term2 (expt diff-j 2))))
              (setq new-val (+ current-val       (* term2 diff-j diff-k))))
           ;; Aggiorna l'elemento k-esimo della riga j-esima
           (setq H-row-new (subst new-val (nth k H-row-new) H-row-new))
         )
         ;; Aggiorna la riga j-esima della matrice H
         (setq H (subst H-row-new H-row H))
       ))))
  H)
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun determinant-3x3 (A / a11 a12 a13 a21 a22 a23 a31 a32 a33)
  (mapcar 'set '(a11 a12 a13 a21 a22 a23 a31 a32 a33) (apply 'append A))
  (+ (* a11 a22 a33)(- (* a11 a23 a32))
     (* a12 a23 a31)(- (* a12 a21 a33))
     (* a13 a21 a32)(- (* a13 a22 a31))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun solve-system-3x3 (A b / r1 r2 r3 a11 a12 a13 a21 a22 a23 a31 a32 a33 b1 b2 b3 det-A)
  (mapcar 'set '(a11 a12 a13 a21 a22 a23 a31 a32 a33) (apply 'append A))
  (mapcar 'set '(b1 b2 b3) b)
  (setq det-A (determinant-3x3 A))
  (if (< (abs det-A) 1e-10)
     (vector-scale b 0.01)
     (list (/ (determinant-3x3 (list (list b1 a12 a13) (list b2 a22 a23) (list b3 a32 a33))) det-A)
```

```
        (/ (determinant-3x3 (list (list a11 b1 a13) (list a21 b2 a23) (list a31 b3 a33))) det-A)
        (/ (determinant-3x3 (list (list a11 a12 b1) (list a21 a22 b2) (list a31 a32 b3))) det-A))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun build-optimization-results (Pui Puf LisCen LisRad iter-num converged lerr lambda-param duration NameMethod)
  (list
    (cons 'radius_list    LisRad)
    (cons 'start_point    Pui)
    (cons 'point          Puf)
    (cons 'iterations     iter-num)
    (cons 'start_error    (compute-error       Pui LisCen LisRad))
    (cons 'error          (compute-error       Puf LisCen LisRad))
    (cons 'residuals      (compute-residuals   Puf LisCen LisRad))
    (cons 'converged      converged)
    (cons 'errors_history (reverse lerr))
    (cons 'lambda_final   lambda-param)
    (cons 'duration       duration)
    (cons 'name_method    NameMethod)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
;; Metodi di ottimizzazione
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun optimize-GD (LisCir start-point lay /
                                    start-time duration-time LisCen LisRad point grad iter-num max-iter learning-rate tol errors residuals)
;metodo della discesa del gradiente
  (setq start-time (contatempo nil))
  (mapcar 'set '(LisCen LisRad) (Get_Data_LisCir LisCir))
  (setq max-iter      1000
        learning-rate 0.01
        tol           1e-6
        errors        nil
                      iter-num      0
                      converged     nil
        point         start-point)
  (while (and (< iter-num max-iter) (not converged))
         (setq grad      (gradient-function point LisCen LisRad))
         (if (< (vector-length grad) tol)
            (setq converged T)
            (progn  (setq point     (Somm_2Lis point (vector-scale grad (- learning-rate))))
                    (setq errors    (cons (objective-function point LisCen LisRad) errors))
                    (setq iter-num (1+ iter-num)))))
```

```
  (setq duration-time (contatempo start-time))

  (Draw_Point point lay)
  (build-optimization-results start-point point LisCen LisRad iter-num converged errors nil duration-time "Discesa Gradiente   "))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun optimize-LM (LisCir start-point lay /
                                    start-time duration-time LisCen LisRad point lambda-param lambda-factor iter-num max-iter tol errors H-mod step residuals)  ; algoritmo di Levenberg-Marquardt
  (setq start-time (contatempo nil))
  (mapcar 'set '(LisCen LisRad) (Get_Data_LisCir LisCir))
  (setq max-iter 1000
        tol 1e-6
        errors       nil
        iter-num     0
        converged    nil
       point         start-point
        lambda-init 0.01
        lambda-factor 10.0
        lambda-param lambda-init)

  (while (and (< iter-num max-iter) (not converged))
        (setq current-error (objective-function point LisCen LisRad)
             errors        (cons current-error errors))
        (setq grad          (gradient-function point LisCen LisRad)
             H             (hessian-function  point LisCen LisRad))
        (setq grad-norm     (vector-length grad))

        (if (< grad-norm tol)
           (setq converged T)
           (progn   (setq H-mod     (matrix-sum H (matrix-scale (matrix-identity 3) lambda-param)))
                    (setq step      (vector-scale grad (/ -1.0 (+ lambda-param 1.0))))
                    (setq new-point (Somm_2Lis point step)
                          new-error (objective-function new-point LisCen LisRad))
                    (if (< new-error current-error)
                       (setq lambda-param (/ lambda-param lambda-factor)
                                                point        new-point)
                       (setq lambda-param (* lambda-param lambda-factor)))
                    (setq iter-num (1+ iter-num)))))
```

```
  (setq duration-time (contatempo start-time))
  (Draw_Point point lay)
  (build-optimization-results start-point point LisCen LisRad iter-num converged errors lambda-param duration-time "Levenberg-Marquardt
"))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun optimize-NR (LisCir start-point lay / start-time duration-time LisCen LisRad point iter-num max-iter tol errors H step residuals)
;metodo di Newton-Raphson
  (setq start-time (contatempo nil))
  (mapcar 'set '(LisCen LisRad) (Get_Data_LisCir LisCir))
  (setq max-iter  1000
        tol       1e-6
        errors    nil
        iter-num  0
        converged nil
      point     start-point)
  (while (and (< iter-num max-iter) (not converged))
    (setq current-error (objective-function point LisCen LisRad)
          errors        (cons current-error errors))
    (setq grad (gradient-function point LisCen LisRad)
          H    (hessian-function  point LisCen LisRad))
    (if (< (vector-length grad) tol)
        (setq converged T)
        (progn
          (setq grad-scaled (vector-scale grad -1.0)
                det-H       (determinant-3x3 H))
          (if (< (abs det-H) 1e-10)
              (setq step (vector-scale grad -0.01))
              (setq step (solve-system-3x3 H grad-scaled)))
          (setq new-point (Somm_2Lis point step)
                new-error (objective-function new-point LisCen LisRad))
          (if (< new-error current-error)
              (setq point new-point)
              (setq point (Somm_2Lis point (vector-scale step 0.5))))
          (setq iter-num (1+ iter-num)))))
  (setq duration-time (contatempo start-time))
  (Draw_Point point lay)
  (build-optimization-results start-point point LisCen LisRad iter-num converged errors nil duration-time "Newton-Raphson      "))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
(defun P_tol (Pu_min Pu_max tol)
  (and (< (abs (- (car   Pu_min) (car   Pu_max))) tol)
       (< (abs (- (cadr  Pu_min) (cadr  Pu_max))) tol)
       (< (abs (- (caddr Pu_min) (caddr Pu_max))) tol)
       (< (distance Pu_min Pu_max)              tol)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun optimize-GA (LisCir lay / Pu_min Pu_max interv apert cicli tol start-time duration-time PurA PurB lis converged cycle-count)  ;metodo con griglia adattiva
  (setq start-time (contatempo nil))
  (mapcar 'set '(LisCen LisRad) (Get_Data_LisCir LisCir))
  (setq interv   32
        apert    2
        cicli    40
        tol    1e-6)
  (mapcar 'set '(PurA PurB)    (Rect_Ini                          LisCir))
  (mapcar 'set '(Pu_min Pu_max) (calc_grid_err    PurA PurB (* 2 interv) LisCir))
  (setq Pu_max_0  Pu_max
        num_cicli 1     )
  (while (and (not (setq converged (P_tol Pu_min Pu_max tol)))
                                  (< num_cicli cicli))
       (progn  (mapcar 'set '(PurA PurB)    (Rect_Gen Pu_min PurA PurB apert interv       ))
               (mapcar 'set '(Pu_min Pu_max) (calc_grid_err   PurA PurB       interv LisCir))
               (setq num_cicli (1+ num_cicli))))
  (setq point Pu_min)
  (setq duration-time (contatempo start-time))
  (Draw_Point point lay)
  (build-optimization-results Pu_max_0 point LisCen LisRad num_cicli converged nil nil duration-time "Griglia Adattiva    "))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun select-three-circles ( / LisCir ent done)
  (while (and (not done) (< (length LisCir) 3))
       (setq ent (car (entsel (strcat "\nSeleziona cerchio " (itoa (1+ (length LisCir))) " di 3: "))))
       (cond  ((null ent)   (if (> (length LisCir) 0)
                            (progn  (princ "\nSelezione annullata.")
                                    (setq LisCir nil)
                                    (setq done T))
                                                (progn  (setq done T))))
              (T            (if (= (Get_DxfVal_Ent ent 0) "CIRCLE")
                            (progn  (setq LisCir (append LisCir (list ent)))
                                    (princ (vl-princ-to-string ent)))
```

```
                    (progn  (princ "\nSeleziona solo entità di tipo cerchio."))))))
  LisCir)
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun princ-results-row (all-results / name_method start_point start_error point error)
  (foreach result all-results
    (mapcar 'set '(name_method start_point start_error point error)  (Get_LisVal_LisAssoc result  '(name_method start_point start_error point error)))
    (princ (strcat "\n" name_method "  "
                   (format-point start_point nil) "  "    (rtos-pad start_error 2  4 6) "    "
                   (format-point point       nil) "  "    (rtos-pad error       2 16 6) "    "))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun princ-best-method (all-results / min-error best-name error name_method)
  (setq min-error 1e99
        best-name nil)
  (foreach result all-results
    (mapcar 'set '(error name_method) (Get_LisVal_LisAssoc result '(error name_method)))
    (if (< error min-error) (setq min-error error best-name name_method)))
  (princ (strcat "\n\nIl metodo con l'errore minore è: " best-name "      con errore " (rtos-pad min-error 2 10 6))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun princ-methods-with-convergence (all-results / name_method iterations converged d-min d-sec)
  (foreach result-pair all-results
    (mapcar 'set '(name_method iterations converged duration)  (Get_LisVal_LisAssoc (cdr result-pair)    '(name_method iterations converged duration)))
    (setq d-min (fix (/ duration 60))
          d-sec (rem duration 60))
    (princ (strcat  "\n" name_method "    convergenza raggiunta: " (if converged "SI" "NO")
                    " - durata calcolo: "  (if (= d-min 0) "" (strcat (itoa d-min) " minuti e "))
                    (rtos-pad d-sec 2 0 2)
                    " secondi - numero di iterazioni: "
                    (itoa-pad iterations 4)))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun princ-radius (all-results / radius-list radius-string avg-radius)
  (mapcar 'set '(radius-list) (Get_LisVal_LisAssoc (car all-results) '(radius_list)))
  (setq avg-radius (/ (apply '+ radius-list) (length radius-list)))
  (setq radius-string "")
  (foreach radius radius-list    (setq radius-string (strcat radius-string (rtos radius 2 6) "      "))  )
  (princ (strcat "\n                          " radius-string "    -   " (rtos avg-radius 2 6))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun princ-residuals-by-method (all-results / name_method residuals avg-abs-residual)
```

```
  (foreach result all-results
    (mapcar 'set '(name_method residuals) (Get_LisVal_LisAssoc result   '(name_method residuals)))
    (setq avg-abs-residual (/ (apply '+ (mapcar 'abs residuals)) (length residuals)))
    (princ (strcat "\n" name_method "    " (rtos-pad (nth 0 residuals) 2 8 4)                "   "
                                  (rtos-pad (nth 1 residuals) 2 8 4)                "   "
                                  (rtos-pad (nth 2 residuals) 2 8 4)                "   -   "
                                  (rtos-pad avg-abs-residual  2 8 4)))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun ----- (num / sep-string) (setq sep-string "\n")  (repeat num (setq sep-string (strcat sep-string "-----"))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun ====== (num / sep-string) (setq sep-string "\n")  (repeat num (setq sep-string (strcat sep-string "======"))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun princ-advanced-results (all-results / best-method)   ;Mostra un riepilogo dei risultati delle ottimizzazioni con 11 punti

  (princ (====== 24))  (princ "\n                          COMPARAZIONE DEI RISULTATI DELLE TRE OTTIMIZZAZIONI")
  (princ (====== 24))   (princ "\nMETODO                    PUNTO DI PARTENZA          ERRORE                PUNTO CALCOLATO
ERRORE")
  (princ (----- 24))  (princ-results-row            all-results)
  (princ (----- 24))  (princ-methods-with-convergence all-results)
  (princ (----- 24))  (princ-best-method             all-results)
  (princ (----- 24))  (princ "\n\nMETODO               RAGGIO 1      RAGGIO 2      RAGGIO 3   -   RAGGIO MEDIO")
  (princ (----- 24))  (princ-radius                          all-results)
  (princ (----- 24))  (princ "\n\nMETODO               ERRORE 1      ERRORE 2      ERRORE 3   -   ERRORE MEDIO")
  (princ (----- 24))  (princ-residuals-by-method     all-results)
  (princ (====== 24)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun calc_grid_err (rect-min rect-max divisions LisCir     /
                                    dx dy x-range y-range min-error max-error Pu_min Pu_max all-spline-rows all-spline-cols)
  (setq dx (/ (- (car  rect-max) (car  rect-min)) divisions))
  (setq dy (/ (- (cadr rect-max) (cadr rect-min)) divisions))
  (setq x-range (mapcar (function (lambda (i) (+ (car  rect-min) (* i dx))))  (range 0 divisions)))
  (setq y-range (mapcar (function (lambda (i) (+ (cadr rect-min) (* i dy))))  (range 0 divisions)))
  (setq min-error  1e99)
  (setq max-error -1e99)
  (mapcar 'set '(LisCen LisRad) (Get_Data_LisCir LisCir))
  (foreach CxGen x-range
    (foreach CyGen y-range
      (setq error (compute-error (list CxGen CyGen 0.0) LisCen LisRad))
      (cond ((< error min-error) (setq min-error error Pu_min (list CxGen CyGen (- error))))
```

```
        ((> error max-error) (setq max-error error Pu_max (list CxGen CyGen (- error)))))))
  (list Pu_min Pu_max))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun NCorMin_LisPu    (LisPu n)       (apply 'min (mapcar (function (lambda (p) (nth n p))) LisPu)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun NCorMax_LisPu    (LisPu n)       (apply 'max (mapcar (function (lambda (p) (nth n p))) LisPu)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PuMin_LisPu      (LisPu)         (mapcar (function (lambda (n) (NCorMin_LisPu LisPu n))) '(0 1 2)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PuMax_LisPu      (LisPu)         (mapcar (function (lambda (n) (NCorMax_LisPu LisPu n))) '(0 1 2)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun LisPuBox_LisPu   (LisPu)         (list (PuMin_LisPu LisPu) (PuMax_LisPu LisPu)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun LisPuQuad_Cir    (cir / cen rad)  (mapcar 'set '(cen rad) (Get_Data_Cir cir)) (mapcar (function (lambda (ang) (polar cen ang rad))) (list
0 pi (/ pi 2) (* 3 (/ pi 2)))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Rect_Ini (LisCir)            (LisPuBox_LisPu (apply 'append (mapcar 'LisPuQuad_Cir LisCir))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Rect_Gen (Pu-min rect-min rect-max multipl divisions)
  (list   (list (- (car  Pu-min) (* multipl (/ (- (car  rect-max) (car  rect-min)) divisions)))
            (- (cadr Pu-min) (* multipl (/ (- (cadr rect-max) (cadr rect-min)) divisions))))
        (list (+ (car  Pu-min) (* multipl (/ (- (car  rect-max) (car  rect-min)) divisions)))
            (+ (cadr Pu-min) (* multipl (/ (- (cadr rect-max) (cadr rect-min)) divisions))))))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun contatempo (st_ti / th_ti du_ti)
   (setq th_ti (getvar "DATE"))
   (if st_ti (progn (setq du_ti (* (- th_ti st_ti) 86400.0)) ;; Differenza in secondi
                    (princ (strcat "\nDurata esecuzione: "  (itoa (fix (/ du_ti 60)))  " minuti"
                                                    " e "               (rtos (rem   du_ti 60) 2 3) " secondi"))
                                    du_ti)
                th_ti))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun C:TRILATCOMPARE (/ start-time duration-time LisCir barycenter  result-gd-b result-lm-b result-nr-b all-results LisCen LisRad)
  (setq start-time (contatempo nil))
  (setvar "CMDECHO" 0)
  (princ "\nTRILATERAZIONE COMPARATIVA AVANZATA - Confronto tra tre ottimizzazioni")
  (setq LisCir (select-three-circles))
  (if (= (length LisCir) 3)
     (progn (mapcar 'Lay_Make_Col '("TRILAT-GD"   "TRILAT-LM"  "TRILAT-NR" "TRILAT-GA") '(1 4 5 6))
```

```
                        (setq barycenter (Media_lis_Pu (car (Get_Data_LisCir LisCir))))
    (princ "\nEsecuzione ottimizzazioni in corso...")
    (setq result-gd (optimize-GD LisCir barycenter "TRILAT-GD"))
    (setq result-lm (optimize-LM LisCir barycenter "TRILAT-LM"))
    (setq result-nr (optimize-NR LisCir barycenter "TRILAT-NR"))
                        (setq result-ga (optimize-GA LisCir          "TRILAT-GA"))
    (princ-advanced-results (list result-gd result-lm result-nr result-ga))))
 (setq duration-time (contatempo start-time))
 (princ))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

CODICE DEL LISTATO 2

```
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(vl-load-com)

(setvar "cmdecho" 0)

(setvar "pdmode" 34)

(setvar "pdsize" -2.5)

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun P_Layer (name) (tblsearch "LAYER" name))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Lay_Make_Col   (lay color)  (if (not (P_Layer lay))  (command "._-layer" "_m" lay "_c" (if (and color (> color 0)) color 7) lay "")

                                        (command "._-layer"          "_c" (if (and color (> color 0)) color 7) lay ""))    lay)

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Lay_Check        (lay) (if (not (P_Layer lay)) (Lay_Make_Col lay 7)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun cons_0   (cod)  (cond  ((= cod 1) (cons 0 "POLYLINE"))    ((= cod 2) (cons 0 "VERTEX"))   ((= cod 3) (cons 0 "SEQEND"))

                      ((= cod 4) (cons 0 "LWPOLYLINE"))  ((= cod 5) (cons 0 "LINE"))     ((= cod 6) (cons 0 "CIRCLE"))

                      ((= cod 7) (cons 0 "POINT"))       ((= cod 8) (cons 0 "SPLINE"))   ((= cod 9) (cons 0 "ELLIPSE"))))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun cons_8   (lay) (cons 8 lay))

(defun cons_11  (Pu1) (cons 11 (Pu_3D Pu1)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun cons_100 (cod)  (cond ((= cod 1) (cons 100 "AcDbEntity"))       ((= cod 2) (cons 100 "AcDb3dPolyline"))

                      ((= cod 3) (cons 100 "AcDbVertex"))       ((= cod 4) (cons 100 "AcDb3dPolylineVertex"))

                      ((= cod 5) (cons 100 "AcDbPolyline"))     ((= cod 6) (cons 100 "AcDbSpline"))

                                        ((= cod 7) (cons 100 "AcDbEllipse"))                                    ))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Draw_Spline (lPu lay clo grado / spl_ent) (Lay_Check lay)      ;Grado 1,2,3,...,< n° Pu: spezzata (non curva), curva “quadratica”, curva “cubica”

  (if (and (> (length lPu) 1)(> (length lPu) grado))

    (setq spl_ent (entmakex   (append (list (cons_0 8) (cons_100 1) (cons_100 6) (cons 70 (if clo 1 0)) (cons_8 lay) (cons 71 grado) (cons 72 (length lPu)))

                              (mapcar '(lambda (Pu) (cons 11 Pu)) lPu))))))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun P_Pu    (Pu) (and (listp Pu) (member (length Pu) '(2 3)) (vl-every 'numberp Pu)))

(defun P_Pu_2D (Pu) (and (P_Pu Pu) (= (length Pu) 2)))

(defun P_Pu_3D (Pu) (and (P_Pu Pu) (= (length Pu) 3)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

(defun Pu_3D          (Pu)        (cond ((P_Pu_2D Pu) (append Pu '(0.0))) ((P_Pu_3D Pu) Pu)))

;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
(defun Ent_Erase (ent)        (if (= (type ent) 'ENAME)  (entdel ent)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_ent (x) (and x (= (type x) 'ENAME)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun contatempo (st_ti / th_ti du_ti)
   (setq th_ti (getvar "DATE"))
   (if st_ti (progn (setq du_ti (* (- th_ti st_ti) 86400.0)) ;; Differenza in secondi
                    (princ (strcat "\nDurata esecuzione: "  (itoa (fix (/ du_ti 60)))  " minuti"
                                                            " e "                (rtos (rem   du_ti 60) 2 3) " secondi"))
                                                    du_ti)
                th_ti))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun #----# (num) (princ (strcat "\n" (if num (rtos num 2 0) "--") (if (and num (< num 10)) "-" "")
                                                                    "-------------------------------------------------------------------------------------------------------"
                                                                          (if (and num (< num 10)) "-" "") (if num (rtos num 2 0) "--"))))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Get_LisVal_LisAssoc    (LisAssoc Liskey)  (mapcar '(lambda (key) (cdr (assoc key LisAssoc))) Liskey))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun P_ent (x) (and x (= (type x) 'ENAME)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Get_DxfVal_Ent           (ent code)  (if (P_ent ent)    (car (Get_LisVal_LisAssoc (entget ent) (list code)))  ))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Get_LisCir (NumCir / LisCir ent done)
  (while (and (not done) (< (length LisCir) NumCir))
       (setq ent (car (entsel (strcat "\nSeleziona cerchio " (itoa (1+ (length LisCir))) " di " (itoa NumCir)  ": "))))
       (cond  ((null ent)   (if (> (length LisCir) 0)
                            (progn  (princ "\nSelezione annullata.")
                                 (setq LisCir nil)
                                 (setq done T))
                                                 (progn  (setq done T))))
              (T           (if (= (Get_DxfVal_Ent ent 0) "CIRCLE")
                            (progn  (setq LisCir (append LisCir (list ent)))
                                 (princ (vl-princ-to-string ent)))
                            (progn  (princ "\nSeleziona solo entità di tipo cerchio."))))))
  LisCir)
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun MS (lst len) (/ (apply '+ lst) len))                                        ; Media Semplice
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
(defun SQM (lst len / m q)                                    ; calcola lo scarto quadratico medio o deviazione standard
  (setq m (/ (apply '+ lst) len))
  (setq q (apply '+ (mapcar '(lambda (x) (* (- x m) (- x m))) lst)))
  (sqrt (/ q len)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun EQMR (lst len) (sqrt (/ (apply '+ (mapcar '(lambda (d) (* d d)) lst)) len)))          ; calcola la Radice quadrata dell'errore quadratico medio o Root Mean Square Error RMSE
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun ICES (lst len) (sqrt (* (EQMR lst len) (SQM lst len))))                    ; indice composto tra errore e scarto
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Test_Pu_LisCir (LisCir Pu1 tipo / LisDis n)
  (setq LisDis (mapcar '(lambda (cir) (distance Pu1 (vlax-curve-getClosestPointTo cir Pu1))) LisCir))
  (setq n (length LisDis))
  (if (> n 0)
     (cond ((= tipo 1) (MS   LisDis n))
           ((= tipo 2) (EQMR LisDis n))
           ((= tipo 3) (SQM  LisDis n))
           ((= tipo 4) (ICES LisDis n)))))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun range (a b / out)  (setq out nil)  (while (<= a b) (setq out (cons a out) a (1+ a)))  (reverse out))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun NCorMin_LisPu    (LisPu n)       (apply 'min (mapcar (function (lambda (p) (nth n p))) LisPu)))
(defun NCorMax_LisPu    (LisPu n)       (apply 'max (mapcar (function (lambda (p) (nth n p))) LisPu)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun PuMin_LisPu      (LisPu)         (mapcar (function (lambda (n) (NCorMin_LisPu LisPu n))) '(0 1 2)))
(defun PuMax_LisPu      (LisPu)         (mapcar (function (lambda (n) (NCorMAX_LisPu LisPu n))) '(0 1 2)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun LisPuBox_LisPu   (LisPu)         (list (PuMin_LisPu LisPu) (PuMax_LisPu LisPu)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Get_BouBox_Ent    (ent)    (if ent    (progn (vla-GetBoundingBox (vlax-ename->vla-object ent) 'Pu_min 'Pu_max) (LisPuBox_LisPu (mapcar 'vlax-safearray->list (list Pu_min Pu_max))))))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Get_BouBox_LisEnt (LisEnt)  (if LisEnt (LisPuBox_LisPu  (apply 'append (mapcar 'Get_BouBox_Ent LisEnt)))))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Build_Grid (LisCir dens tipo / LisPu minX maxX minY maxY minZ maxZ  dx dy grid Cx Cy)
  (setq LisNum  (range 0 (- dens 1)))
  (setq LisPu   (Get_BouBox_LisEnt LisCir))
  (mapcar 'set '(minX minY minZ maxX maxY maxZ) (apply 'append LisPu))
  (setq dx (/ (- maxX minX) (- dens 1.0)))
```

```
  (setq dy (/ (- maxY minY) (- dens 1.0)))
  (setq grid  (mapcar (function (lambda (i)
                                (mapcar (function (lambda (j)  (setq Cx (+ minX (* dx j)))
                                                               (setq Cy (+ minY (* dy i)))
                                                               (list Cx Cy (- (Test_Pu_LisCir LisCir (list Cx Cy 0.0) tipo))))) LisNum)))
LisNum)))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_Spl_From_Grid (rmse_grid lay grado / x_splines y_splines col i)
  (setq x_splines nil y_splines nil)
  (foreach row rmse_grid (setq x_splines (cons (Draw_Spline row lay nil grado) x_splines)))
  (setq x_splines (reverse x_splines))
  (setq i 0)
  (while (< i (length (car rmse_grid)))
        (setq col nil)
        (foreach row rmse_grid (setq col (append col (list (nth i row)))))
        (setq y_splines (cons (Draw_Spline col lay nil grado) y_splines))
        (setq i (1+ i)))
  (setq y_splines (reverse y_splines))
  (list x_splines y_splines))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_SurNet_From_Spl (spline_lists lay del_flag / old_layer xlist ylist surf_ent)
  (setq old_layer (getvar "clayer"))
  (Lay_Check lay)
  (setvar "clayer" lay)
  (setq xlist (car spline_lists))
  (setq ylist (cadr spline_lists))
  (apply 'command-s (append (list "._surfnetwork") xlist (list "") ylist (list "")))
  (setq surf_ent (entlast))
  (if del_flag  (progn (foreach ent xlist (Ent_Erase ent))
                       (foreach ent ylist (Ent_Erase ent))))
  (setvar "clayer"   old_layer)
  surf_ent)
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_Box_LisPu (LisPu lay / old_layer Pu1 Pu2)
  (setq old_layer (getvar "clayer"))
  (Lay_Check lay)
  (setvar "clayer" lay)
  (mapcar 'set '(Pu1 Pu2) LisPu)
  (command "._BOX" "_non" Pu1 "_non" Pu2)
```

```
  (setvar "clayer"   old_layer)
  (entlast))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_Sol_From_Surnet (surf_Ent LisPuBox lay / old_osmode old_3dosmode box_Ent1 box_Ent2 c1 c2 z1 z2)
  (mapcar 'set '(Pu1 Pu2)  LisPuBox)
  (setq box_Ent   (Draw_Box_LisPu (list Pu1 (list (car Pu2) (cadr Pu2) 0)) lay))
  (if (and box_Ent surf_Ent)
    (progn (setq old_osmode   (getvar "osmode"))   (setvar "osmode" 16384)
          (setq old_3dosmode (getvar "3dosmode")) (setvar "3dosmode"   1)
          (command "._SLICE" box_Ent "" "_Surface" surf_Ent "_B")
          (setvar "osmode"   old_osmode)
          (setvar "3dosmode" old_3dosmode)
          (setq box_Ent1 box_Ent)
          (setq box_Ent2 (entlast))
          (setq z1 (caddr (vlax-get (vlax-ename->vla-object box_Ent1) 'Centroid)))
          (setq z2 (caddr (vlax-get (vlax-ename->vla-object box_Ent2) 'Centroid)))
          (if (< z1 z2)
             (progn (entdel box_Ent2) box_Ent1)
             (progn (entdel box_Ent1) box_Ent2)))))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Draw_Sections_From_Solid (sol_ent LisCz num_sez lay / vla_sol_ent stepZ i z)
  (setq old_layer (getvar "clayer"))
  (Lay_Check lay)
  (setvar "clayer" lay)
  (mapcar 'set '(minZ maxZ) LisCz)
  (setq vla_sol_ent (vlax-ename->vla-object sol_ent))
  (setq stepZ (/ (- maxZ minZ) (float num_sez)))
  (setq i 0)
  (while (<= i num_sez)
    (setq z (+ minZ (* stepZ i)))
    (vla-SectionSolid vla_sol_ent (vlax-3d-point 0   0  z) (vlax-3d-point 100 0  z) (vlax-3d-point 0   100 z))
    (setq i (1+ i)))
  (setvar "clayer"   old_layer)
  (princ))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun LisDec_Num (num / out val)
  (setq val 1)
  (setq num (abs num))
```

```
  (while (<= val num)
        (setq out (cons val out))
        (setq val (* val 10)))
  (reverse (cons val out)))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun OrdineDimensione (num / decadi ordine absnum)
  (setq decadi (LisDec_Num num))
  (setq num    (abs num))
  (setq ordine (car decadi))
  (foreach d decadi (setq ordine (if (<= d num) d ordine))))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun OrdineEQuantita (num / ordine quantita)
  (setq ordine (OrdineDimensione num))
  (setq quantita (1+ (fix (/ (abs num) ordine))))
  (list ordine quantita))
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Sol_Analysis (LisCir dens fuob DrawSez lay1 lay2 lay3 lay4  /
                                        dens old_osmode old_3dosmode LisPuGrid LisSplEnt SurnetEnt SurnetBox SolboxEnt SolsurEnt)
  (setq old_osmode   (getvar "osmode"))   (setvar "osmode" 16384)
  (setq old_3dosmode (getvar "3dosmode")) (setvar "3dosmode"    1)
  (setq dens (min (max dens 3) 9)); limitatore di sicurezza per prevenire blocchi nella  generazione del solido
  (setq LisPuGrid   (Build_Grid           LisCir (expt 2 dens) fuob))    ;fuob 2 SQM  3 deviazione standard 4 mix
  (setq LisPuBox    (LisPuBox_LisPu       (apply 'append LisPuGrid)))
  (setq LisSplEnt   (Draw_Spl_From_Grid    LisPuGrid "0" 3))  ; "0" tanto le butto 3 grado della spline
  (setq SurnetEnt   (Draw_SurNet_From_Spl  LisSplEnt lay1 T)) ; T butto le
  (setq SolsurEnt   (Draw_Sol_From_Surnet  SurnetEnt LisPuBox lay2))
  (if DrawSez (progn (setq CzMin_OrdQua (OrdineEQuantita (caddr (car  LisPuBox))))
                                        (setq CzMax_OrdQua (OrdineEQuantita (caddr (cadr LisPuBox))))
                                        (setq rangeMin  (list (- (apply '* CzMin_OrdQua)) 0))
                                        (setq rangeMax  (list (- (apply '* CzMax_OrdQua)) 0))
                (setq numsezMin (* 10  (cadr CzMin_OrdQua)))
                                        (setq numsezMax (* 10  (cadr CzMax_OrdQua)))
                                        (Draw_Sections_From_Solid SolsurEnt rangeMin numsezMin lay3)
                                        (Draw_Sections_From_Solid SolsurEnt rangeMax numsezMax lay4)))
  (setvar "osmode"   old_osmode)
  (setvar "3dosmode" old_3dosmode))
;(Draw_Sections_From_Solid (car(entsel))'(-100 0) 100 "sezioni 1.0")
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
(defun Get_Tipo_Analisi (/ val)
  (initget "MS EQMR SQM ICES")
  (setq val (getkword "\nScegli tipo di analisi [MS/EQMR/SQM/ICES] <EQMR>: "))
  (cond ((= val "MS")   1)
        ((= val "EQMR") 2)
        ((= val "SQM")  3)
        ((= val "ICES") 4)
        ((null val)     2)   ; default  EQMR
        (T              2))) ; fallback EQMR
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun Get_Densita_Griglia (/ val)
  ;; Stampa tabella informativa
  (princ "\nLa densità della griglia viene calcolata come potenza ennesima di 2.")
  (princ "\nI punti della griglia sono determinati dal quadrato della densità della griglia.")
  (princ "\n+---+------------------+---------------+")
  (princ "\n| n | Densita' Griglia | Punti Griglia |")
  (princ "\n+---+------------------+---------------+")
  (princ "\n| 4 |     16 x 16      |       256     |")
  (princ "\n| 5 |     32 x 32      |     1.024     |")
  (princ "\n| 6 |     64 x 64      |     4.096     |")
  (princ "\n| 7 |    128 x 128     |    16.384     |")
  (princ "\n| 8 |    256 x 256     |    65.536     |")
  (princ "\n+---+------------------+---------------+")
  (while (or (null val) (not (member val '(4 5 6 7 8))))
    (setq val (getint "\nScegli n per densità e punti della griglia [ 4 5 6 7 8 ] <6>: "))
    (if (null val) (setq val 6))  ; default 6 (64x64)
    (if (not (member val '(4 5 6 7 8)))
        (prompt "\nValore non valido. Inserisci un numero tra 4 e 8.")))
  val)
;;-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
(defun c:analisi ( / old_osmode old_3dosmode start-time duration-time LisCir supnet tipo_analisi dens_val nome_tipo lay_sup lay_sol lay_sez_min lay_sez_max)
  (setq old_osmode   (getvar "osmode"))   (setvar "osmode" 16384)
  (setq old_3dosmode (getvar "3dosmode")) (setvar "3dosmode"    1)
  (setq start-time (contatempo nil))
  (#----# 1)  (setq tipo_analisi (Get_Tipo_Analisi))
  ;; Determina il nome del tipo per i layer
  (setq nome_tipo (cond ((= tipo_analisi 1) "MS")
                        ((= tipo_analisi 2) "EQMR")
```

```
                ((= tipo_analisi 3) "SQM")
                ((= tipo_analisi 4) "ICES")))
  (princ (strcat "\nTipo analisi: " nome_tipo))
  (#----# 2)  (setq dens_val      (Get_Densita_Griglia))
  (princ (strcat "\nDensita' griglia: " (itoa (expt 2 dens_val)) "x" (itoa (expt 2 dens_val)) " punti"))
  (#----# 3)
  (setq lay_superf (strcat "Tril_" nome_tipo "_Superf"))
  (setq lay_solido (strcat "Tril_" nome_tipo "_Solido"))
  (setq lay_sezcom (strcat "Tril_" nome_tipo "_SezCom"))
  (setq lay_sezapi (strcat "Tril_" nome_tipo "_SezApi"))
  ;; Creazione layer con colori
  (mapcar 'Lay_Make_Col (list lay_superf lay_solido lay_sezcom lay_sezapi) '(40 1 9 6))
  (princ "\nLAYER CHE VERRANNO UTILIZZATI")
  (princ (strcat "\n  Superficie ..... : " lay_superf    " (colore 40 - arancione)"))
  (princ (strcat "\n  Solido ..... : " lay_solido    " (colore  1 - rosso)"))
  (princ (strcat "\n  Sezioni complessive .. : " lay_sezcom    " (colore  9 - grigio chiaro)"))
  (princ (strcat "\n  Sezioni apicali ..... : " lay_sezapi    " (colore  6 - magenta)"))
  (#----# 4)  (setq LisCir  (Get_LisCir 3))
  (#----# 5)  (setq supnet  (Sol_Analysis LisCir dens_val tipo_analisi T lay_superf lay_solido lay_sezcom lay_sezapi))
          (setq duration-time (contatempo start-time))
          (setvar "osmode"   old_osmode)
          (setvar "3dosmode" old_3dosmode)
  (#----# 6)  (princ))
;;-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```