# Manuale codice “VATT DMST”

Questo documento illustrerà brevemente il funzionamento e la struttura del codice MATLAB per la valutazione di performance di una turbina marina ad asse verticale (VATT) mediante teoria DMST con aggiunta di sottomodelli per tener conto di fenomeni idrodinamici ulteriori.

Per la teoria del modello, si rimanda al lavoro di tesi di Stefano Deluca scaricabile sul portale ETD dell’Università di Pisa ([Link](https://etd.adm.unipi.it/theses/browse/by_type/LM.html)).

Tutte le unità adottate in questo lavoro sono sempre SI, *m* per le lunghezze, *s* per il tempo, *kg* per la massa.

Si consiglia di leggere il manuale nella sua interezza prima di effettuare simulazioni, possibilmente visionando il codice sorgente allo stesso tempo.

# Prerequisiti

Il programma è stato sviluppato e testato su sistema operativo **Windows 10 (64 bit)**. L’uso su altre piattaforme potrebbe richiedere modifiche al codice.

È necessario installare MATLAB R2018b con pacchetti:

* MATLAB Coder 4.1
* Parallel Computing Toolbox 6.13
* Image Processing Toolbox 10.3 (solo per simulazioni MIT)

Successivamente bisogna installare:

* MATLAB Support for MinGW-w64 C/C++ Compiler ([Link](https://it.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler))

L’uso di versioni diverse non è stato testato.

# Come lanciare una simulazione normale

Questo paragrafo è volutamente discorsivo. Si rimanda all’analisi dei singoli file per informazioni sulla sintassi degli script.

1. Aprire il file *init\_input.m* e definire le opzioni della simulazione.
2. Lanciare la funzione *dmst\_vatt.m* specificando il profilo di velocità da usare, per simulazioni 3D, oppure semplicemente il valore di velocità del flusso, per simulazioni 2D. Facoltativamente, definire il nome del file di output.
3. Realizzare i plot desiderati usando i dati di output.

# Come lanciare una simulazione con i dati MIT

Al momento sono disponibili due set di dati posizionati in due cartelle.

* Hz600mN010mw (griglia con risoluzione 600 m)
* Hz200mN010mw (griglia con risoluzione 200 m)

Il file con i dati è *out.nc*.

1. Per preparare i dati alla simulazione DMST, avviare lo script *MIT\_read\_nc\_data.m* modificando la variabile pe\_file al suo interno con il percorso al file .nc desiderato. Verrà prodotto, insieme ad altri file, il file *%nome\_file%\_processed.mat.*
2. Per avviare la simulazione MIT, aprire lo script MIT\_run\_simulation.m e modificare la variabile nome\_file con il file *%nome\_file%\_processed.mat* creato precedentemente (senza estensione).
3. Modificare la variabile sim\_step per impostare la risoluzione desiderata.
4. Avviare lo script.
5. I risultati complessivi sono salvati in *%nome\_file%\_sub\_finished.mat.* Sono anche presenti i file delle singole simulazioni in una sottocartella.

# Struttura dei file

In questo paragrafo verranno investigati i singoli script, di cui verranno spiegati i rispettivi input e output. Non si entrerà nei meriti del codice in maniera troppo tecnica in quanto gli script sono sufficientemente commentati. Per informazioni sugli script nella cartella *func*, data la loro natura molto specifica, si consiglia di leggere direttamente il codice sorgente.

### vatt\_dmst.m

Questo file è la function da cui lanciare le simulazioni.

#### Sintassi

[data\_post, data\_geom, data\_vel, data\_out\_geom, data\_out, data\_dyn, sim\_input, sim\_settings] = vatt\_dmst(vel\_input, [output\_file], [tsr\_override]);

|  |  |  |
| --- | --- | --- |
| **Output** | | |
| *Nome* | *Tipo* | *Descrizione* |
| data\_post | Struttura | Contiene vettori utili per l’output |
| data\_geom | Struttura | Contiene grandezze informazioni sulla geometria calcolata del rotore |
| data\_vel | Struttura | Contiene grandezze relative al flusso in ingresso |
| data\_out\_geom | Matrice | Contiene grandezze relative alla geometria del rotore e del flusso calcolate su ogni cella del rotore. Dimensioni: nz\*n\_ring\*16 |
| data\_out | Matrice | Contiene grandezze di output calcolate su ogni cella del rotore. Dimensioni: nz\*n\_ring\*8 |
| data\_dyn | Matrice | Contiene grandezze relative alla routine di stallo dinamico per ogni cella del rotore. È una matrice nz\*n\_ring\*8 |
| sim\_input | Struttura | Contiene impostazioni della simulazione |
| sim\_settings | Struttura | Contiene informazioni sui sottomodelli attivi |

|  |  |  |
| --- | --- | --- |
| **Input** | | |
| *Nome* | *Tipo* | *Descrizione* |
| vel\_input | Matrice/  Scalare | Contiene informazioni sul flusso indisturbato.   * *Simulazioni 2D*   Scalare della velocità.   * *Simulazioni 3D*   Matrice di 2 colonne.  La prima indica le posizioni z a cui la velocità viene misurata da 0 (pelo libero del mare) a (fondale). La seconda, il valore di velocità. |
| output\_file | Stringa | Opzionale. Indica il file di output dove salvare le simulazioni. Se non fornito, viene generato un nome dal programma. |
| tsr\_override | Scalare | Opzionale. Imposta un TSR diverso da quello specificato in init\_input.m |

Per un esempio su come avviare una simulazione, fare riferimento allo script *ESEMPIO\_lanciasim\_singola.m* e *ESEMPIO\_dmst\_optimizer.m*

I nomi delle variabili all’interno delle strutture sono piuttosto intuitivi. In caso di dubbio, risalire dal codice sorgente alla grandezza calcolata.

Ora verranno illustrate le grandezze calcolate nelle matrici, riferite alla cella in posizione azimutale *i*, variabile fra 1 e n\_ring, e posizione verticale *k*, variabile fra 1 e nz.

|  |  |  |  |
| --- | --- | --- | --- |
| **data\_out(k,i,x)** | | | |
| 1 | Fattore di induzione assiale non corretto | 5 | Fattore di perdita alle pute |
| 2 | Fattore di induzione assiale a monte, se la cella è in downstream | 6 | CL senza perdite alle punte |
| 3 | CL con perdita alle punte | 7 | CD senza perdite alle punte |
| 4 | CD con perdita alle punte | 8 | Fattore di induzione assiale dopo correzione espansione streamtubes |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **data\_out\_geom(k,i,x)** | | | | | |
| 1 |  | Raggio | 9 |  | Coseno dell’angolo di attacco |
| 2 |  | Coseno di pos azimutale | 10 | [deg] | Angolo di attacco |
| 3 |  | Seno di pos azimutale | 11 |  | Angolo di attacco virtuale |
| 4 |  | Componente orizzontale di vel relativa | 12 |  | Altezza della cella |
| 5 |  | Componente verticale di vel relativa | 13 |  | Posizione azimutale |
| 6 |  | Modulo della vel relativa | 14 |  | TSR locale nella cella |
| 7 |  | Numero di Reynolds | 15 |  | Componente orizzontale di vel assoluta |
| 8 |  | Seno dell’angolo di attacco | 16 |  | Componente verticale di vel assoluta |

La definizione della matrice data\_dyn(k,i,x) dipende dal tipo di modello di stallo dinamico adottato. Si invita ad ispezionare *dmst\_calc.m* per maggiori informazioni. Queste informazioni non dovrebbero essere particolarmente rilevanti ai fini dell’output del codice DMST.

### init\_input.m

Questo file è lo script che definisce le impostazioni della simulazione a livello di geometria di turbina, tipo di flusso, discretizzazione del rotore, ecc. Seguire i commenti nel file per configurare la simulazione come si desidera.

#### Sintassi

[sim\_settings, sim\_input] = init\_input

Per gli output si rimanda al paragrafo *vatt\_dmst.m*.

### init\_geom.m

Questo file si occupa di realizzare la discretizzazione spaziale del rotore in nz\*n\_ring celle.

#### Sintassi

[data\_geom] = init\_geom(sim\_input)

Per gli input e output si rimanda al paragrafo *vatt\_dmst.m*.

### init\_vel.m

Questo file si occupa di applicare il flusso indisturbato ad ogni cella del rotore.

#### Sintassi

[data\_vel] = init\_vel(sim\_input, data\_geom, vel\_input)

Per gli input e output si rimanda al paragrafo *vatt\_dmst.m*.

### dmst\_update.m

Questo file gestisce la singola iterazione del solutore DMST. Nello specifico, separa le informazioni disponibili per ogni piano di turbina *k* e successivamente risolve in parallelo i singoli piani di turbina (sezione parfor).

#### Sintassi

[data\_out\_geom, data\_out, data\_dyn] = dmst\_update(sim\_settings, sim\_input, data\_geom, data\_vel)

Per gli input e output si rimanda al paragrafo *vatt\_dmst.m*.

Si sottolinea che la variabile interna dyn\_input è dichiarata come variabile di tipo persistent e pertanto continua ad esistere fra le varie chiamate di *dmst\_update.m.*

### dmst\_par\_loop.m

Questo file gestisce la soluzione del singolo piano di turbina. Viene risolta prima la parte upstream, trovando il fattore di induzione che fa tornare i bilanci per ogni cella azimutale della prima metà di turbina, poi c’è l’eventuale correzione di espansione dei tubi di flusso e successivamente viene risolto il downstream.

#### Sintassi

[geom\_out\_data, out\_data, dyn\_data] = dmst\_par\_loop(sim\_settings, sim\_input, dmst\_input)

|  |  |  |
| --- | --- | --- |
| **Input** | | |
| *Nome* | *Tipo* | *Descrizione* |
| dmst\_input | Struttura | Struttura che contiene informazioni sul singolo piano da risolvere |

Per i restanti input e output si rimanda al paragrafo *vatt\_dmst.m*.

### dmst\_calc.m

Questo file risolve il singolo streamtube per il valore del fattore di induzione fornito.

#### Sintassi

[f\_eff, data\_geom, data\_out, data\_dyn] = dmst\_calc(sim\_settings, sim\_input, dmst\_input, pos\_theta, a, a\_upstream)

|  |  |  |
| --- | --- | --- |
| **Output** | | |
| *Nome* | *Tipo* | *Descrizione* |
| f\_eff | Scalare | Differenza fra forza di spinta calcolata con teoria Actuator Disk e Blade Element |

|  |  |  |
| --- | --- | --- |
| **Input** | | |
| *Nome* | *Tipo* | *Descrizione* |
| pos\_theta | Scalare | Indice della posizione azimutale |
| a | Scalare | Fattore di induzione da adottare |
| a\_upstream | Scalare | Fattore di induzione della cella a monte, se applicabile |

Per i restanti input e output si rimanda al paragrafo *vatt\_dmst.m*.

### dmst\_post.m

Questo file crea vettori utili per l’output

#### Sintassi

post\_data = dmst\_post(sim\_input, out\_geom\_data, geom\_data, out\_data, data\_vel)

Per gli input e output si rimanda al paragrafo *vatt\_dmst.m*.

### dmst\_plot\_update.m

Questo file crea e aggiorna i plot di interesse ad ogni iterazione.

#### Sintassi

dmst\_plot\_update(fig\_out, data\_post)

|  |  |  |
| --- | --- | --- |
| **Input** | | |
| *Nome* | *Tipo* | *Descrizione* |
| fig\_out | Figura | Figura dove creare/aggiornare plots |

Per gli input e output si rimanda al paragrafo *vatt\_dmst.m*.

# Future work

Segue una lista di possibili spunti per il miglioramento del codice, in ordine di crescente difficoltà di implementazione (circa), con relativi commenti.

1. **Cambiare il database di dati di riferimento per profili in accordo con (**[**Link**](https://www.sciencedirect.com/science/article/pii/S1364032114005784)**).**

Bisogna valutare se effettivamente ne valga la pena, ma è molto semplice da fare, bisogna solo modificare *static\_data.m* e il sorgente della routine di stallo dinamico con i nuovi valori.

1. **Raffinare modello di curvatura di streamlines**

Si consiglia di effettuare nuove simulazioni CFD 2D con vario numero di pale, TSR, ecc. In ognuna bisognerà impostare un report per ogni timestep della quantità *Velocity Angle* su una circonferenza centrata nell’origine (dov’è il centro della turbina) di raggio pari al raggio nominale della turbina. Ogni report dovrà essere mediato nel tempo (per usare valori del velocity angle privi di fluttuazioni). L’idea è ricavare un database di velocity angle al variare di posizione azimutale, TSR e numero di pale ed eventualmente altro. Successivamente il fattore dovrà essere ricavato da una interpolazione lineare in più dimensioni (ispirarsi al modello di perdite alle punte nuovo).

1. **Raffinare modello di espansione degli streamtube**

Migliorare la giustificazione fisica del modello corrente o trovarne uno nuovo. Sicuramente migliorare il fitting effettuato sul modello attuale.

1. **Passare ad una discretizzazione verticale non più uniforme, con infittimento verso le punte.**

Potrebbe essere piuttosto ostico, soprattutto perché bisognerebbe cambiare molte formule in giro per il codice.

1. **Implementare effetti di curvatura del flusso.**

Bisogna prima capire perché il modello presente non funziona per codici DMST ma va bene per UDF su Fluent. Se non dovesse andar bene, va reimplementato in altro modo.

1. **Migliorare modello di perdite alle punte**

Servono dati da simulazioni CFD 3D, possibilmente dipendenti anche dalla posizione azimutale. Bisogna capire quanto è importante. Non particolarmente difficile di per sé, ma richiede simulazioni CFD molto lunghe.