Universidade Federal de Uberlândia
Laboratório de Mecânica dos fluidos
Programa de Pós-Graduação em Engenharia Mecânica

# MFSIM:
# MANUAL DE INSTALAÇÃO

Uberlândia
June 13, 2023

# Contents


**1 Introdução** **2**
- 1.1 Obtendo o MFSim . . . 2
  - 1.1.1 Windows . . . 2
  - 1.1.2 Linux . . . 3
- 1.2 Softwares em cada stack . . . 3
- 1.3 Macros . . . 4

**2 Configurando os ambientes** **6**
- 2.1 Windows . . . 6
  - 2.1.1 Docker . . . 6
- 2.2 Linux . . . 7
  - 2.2.1 Singularity . . . 7
    - 2.2.1.1 Instalando o singularity via instalador . . . 7
    - 2.2.1.2 Imagens e versões da stack . . . 7
    - 2.2.1.3 Testando a instalação . . . 8
  - 2.2.2 Lmod . . . 8
    - 2.2.2.1 Instalando a lmod . . . 8
    - 2.2.2.2 Atualizando a lmod . . . 9
- 2.3 Gmsh . . . 9

**3 Filosofia de uso** **10**
- 3.1 Pastas de compilação e Instalação . . . 10
- 3.2 Escolhendo o local de instalação . . . 11
  - 3.2.1 Lmod e Singularity . . . 11
  - 3.2.2 Docker . . . 12
- 3.3 Modos de Instalação . . . 12
- 3.4 Instalação de testes . . . 12

**4 Compilando e Instalando** **13**
- 4.1 Lmod . . . 13
  - 4.1.1 Módulos versão 10 . . . 13
  - 4.1.2 Módulos versão 12 . . . 13
  - 4.1.3 Compilando e instalando . . . 13
- 4.2 Singularity . . . 13
- 4.3 Docker . . . 14

**5 Ferramentas de Apoio** **15**
- 5.1 Ferramenta de Limpeza . . . 15
- 5.2 Ferramenta de Teste de Casos . . . 15

**6 Homologação** **17**
- 6.1 Windows . . . 17
  - 6.1.1 Docker . . . 17
    - 6.1.1.1 Instalação das bibliotecas e dependências . . . 17
    - 6.1.1.2 Obtendo o MFSim e configurando o Docker para uso . . . 17
    - 6.1.1.3 Compilando o MFSim . . . 17
    - 6.1.1.4 Validando . . . 18
- 6.2 Linux . . . 18
  - 6.2.1 Singularity . . . 18
    - 6.2.1.1 Instalação das bibliotecas e dependências . . . 18
    - 6.2.1.2 Obtendo o MFSim e configurando-o para uso . . . 18
    - 6.2.1.3 Compilando o MFSim . . . 18
    - 6.2.1.4 Validando . . . 18
  - 6.2.2 Lmod . . . 19
    - 6.2.2.1 Instalação das bibliotecas e dependências . . . 19
    - 6.2.2.2 Obtendo o MFSim e configurando-o para uso . . . 19
    - 6.2.2.3 Compilando o MFSim . . . 19
    - 6.2.2.4 Validando . . . 19

# 1 Introdução

Antes de utilizar o MFSim é necessário instalar e configurar os softwares que o mesmo depende para funcionar. O conjunto desses softwares é o que chamamos de *stack*. Há 3 *stacks* suportadas pelo MFSim atualmente: 2 para o sistema operacional (SO) GNU/Linux e 1 para o Microsoft Windows. As *stacks* também possuem versões que são nomeadas de acordo com a versão do compilador usado em cada uma delas.

Nesse manual apresentaremos todas as 3 *stacks*, nas duas versões que existem para cada uma. Cabe ao usuário escolher qual a *stack* melhor lhe atende. A seguir está um resumo de cada *stack*, em qual sistema operacional (SO) elas executam, os pontos fortes e fracos de cada uma.

Table 1: Stacks do MFSim

| Stack | Versões | SO | Prós | Contras |
|---|---|---|---|---|
| Docker | 10 e 12 | Windows | instalação mais fácil<br>softwares via imagem pronta | requisitos do Docker |
| Singularity | 10 e 12 | Linux | instalação mais fácil<br>softwares via imagem pronta | requisitos do Singularity |
| Lmod | 10 e 12 | Linux | maior flexibilidade<br>mesmo ambiente dos clusters | instalação mais complexa |

Agora que você sabe os pontos fortes e fracos de cada *stack*, pode então decidir qual delas melhor lhe atende. Evidentemente, algumas stacks são melhor indicadas para alguns ambientes do que outros. Ex: se você pretende utilizar o MFSim no Microsoft Windows, a opção para você é a *stack* Docker.

Na seção 2, mostraremos como instalar e configurar cada uma dessas *stacks*. As subseções estão organizadas por sistema operacional para facilitar, então você pode ir diretamente para a subseção que apresenta a *stack* que você escolheu, sem precisar ler as seções com as demais stacks, sem qualquer prejuízo.

Na seção 3 apresentaremos a filosofia de uso do MFSim. Essa seção **é vital**, então é altamente recomendado que você a leia e procure compreende-la bem, pois precisará dos conhecimentos nela listados posteriormente.

A seção 4 apresenta a compilação e instalação do MFSim. As subseções estão organizadas por *stack*, então, novamente, você pode ir diretamente para a subseção que apresenta como compilar e instalar o MFSim na sua stack.

Em seguida, a seção 5 apresenta as ferramentas de apoio do MFSim e por último, a seção 6 mostra como homologar a instalação da *stack*. Novamente, você pode ir diretamente para a subseção que descreve como homologar a *stack* que você está usando.

## 1.1 Obtendo o MFSim

Para utilizar o MFSim, evidentemente você precisará do código fonte do mesmo.

Atualmente o código fonte é mantido em um servidor privado (gitea) do MFLab. Logo, se ainda não tem acesso ao servidor, precisa providenciar. Você conseguirá obter acesso com o seu professor orientador (se aluno) ou diretamente com a coordenação do MFLab (se externo) pelo e-mail secmflab@mecanica.ufu.br.

Para os próximos passos assumirei que já possui acesso ao gitea. A primeira coisa que precisa fazer é instalar o git, caso já não tenha instalado.

### 1.1.1 Windows

Se você está utilizando Windows você pode fazer o download do git diretamente do site do desenvolvedor: `https://git-scm.com/downloads`.

Baixe e instale o git.

O MFSim é um programa em modo texto (vulgo, linha de comando), então é necessário que você acostume a usar terminais para executar tarefas. O Windows possui um terminal mais sofisticado que o cmd que vem por padrão em toda versão do Windows. Esse terminal é o *PowerShell*. Você pode fazer o download do PowerShell pelo site: `https://docs.microsoft.com/pt-br/powershell/scripting/install/installing-powershell-core-on-windows?view=powershell-7`.

Baixe e instale o powershell.

Feito isso crie um pasta no seu sistema de arquivos. Você pode usar qualquer nome que desejar, porém como utilizará a pasta em linha de comando, é uma boa prática evitar utilizar espaços e acentuação no nome da pasta.

**Para fins de exemplificação** vou considerar que a pasta está em `C:\Users\mflab\Desktop\MFSim`.

Abra o PowerShell e execute:

```
[PowerShell] > cd C:\Users\mflab\Desktop\MFSim
```

Agora que está dentro da pasta, você deve clonar o repositório com o código fonte do MFSim e o repositório com os casos.

```
[PowerShell] > git clone https://www.mflab.mecanica.ufu.br/gitea/root/MFSim-cmake.git
[PowerShell] > git clone https://www.mflab.mecanica.ufu.br/gitea/root/MFSim-cases.git
```

Ao final terá duas novas pastas dentro da que criou originalmente:

```
C:/Users/mflab/Desktop/MFSim
       /MFSim-cmake <- pasta com o código fonte
       /MFSim-cases <- pasta com os casos
```

Pronto, agora você já tem o que precisa para instalar a *stack*.

### 1.1.2 Linux

A maioria das distribuições Linux possui o git disponível para instalação em seus repositórios. Se estiver utilizando Ubuntu ou Mint, você pode instalar o git com o comando:

```
$ sudo apt install git
```

Caso esteja utilizando outra distribuição, adapte o comando conforme o necessário.
Agora que tem o git, crie uma pasta para o MFSim dentro do seu *home*. Para isso abra o terminal e execute:

```
$ cd ~
$ mkdir nome_da_pasta
```

Eu vou utilizar como nome da pasta, `MFSim`, então o comando ficará como:

```
$ cd ~
$ mkdir MFSim
```

Vá para a pasta recém criada

```
$ cd MFSim
```

E clone o repositório com o código fonte do MFSim e o repositório com os casos.

```
$ git clone https://www.mflab.mecanica.ufu.br/gitea/root/MFSim-cmake.git
$ git clone https://www.mflab.mecanica.ufu.br/gitea/root/MFSim-cases.git
```

Ao final terá duas novas pastas dentro da que criou originalmente:

```
$HOME/MFSim
       /MFSim-cmake <- pasta com o código fonte
       /MFSim-cases <- pasta com os casos
```

Pronto, agora você já tem o que precisa para instalar a *stack*.

## 1.2 Softwares em cada stack

A seguir há uma lista dos softwares e versões dos mesmos disponíveis em cada *stack*. A lista é muito mais para fins de informação, visto que não intereferem no uso do MFSim.

Table 2: Softwares - Stacks versão 10

| Software | Docker | Singularity | Lmod |
|---|---|---|---|
| OS | Ubuntu 20.04 [imagem]; Windows [hospedeiro] | CentOS 7 [imagem]; Linux [hospedeiro] | Ubuntu 20.04; Mint 20.4 |
| compilador | gnu 10.2 | gnu 10.2 | gnu 10.2 |
| python | 2.7.18 | 2.7.18 | 2.7.18; 3.9.0 |
| valgrind | 3.16.1 | 3.16.1 | 3.16.1 |
| cmake | 3.19.4 | 3.19.4 | 3.24.1 |
| lapack | 3.9.0 | 3.9.0 | 3.9.0 |
| swig | 3.0.12 | 3.0.12 | 3.0.12 |
| boost | 1.76.0 | 1.76.0 | 1.74.0 |
| eigen3 | 3.3.8 | 3.3.8 | 3.3.8 |
| fmt | 7.1.2 | 7.1.2 | 7.1.2 |
| scons | 3.1.2 | 3.1.2 | 3.0.1 |
| googletest | 1.10.0 | 1.10.0 | 1.10.0 |
| openmpi | 4.0.5 | 4.0.5 | 4.0.5 |
| gsl | 2.7 | 2.7 | 2.7 |
| hdf5 | 1.12.0 | 1.12.0 | 1.10.5 |
| trilinos/zoltan | 13.0.1 | 13.0.1 | 13.0.1 |
| scalapack | 2.1.0 | 2.1.0 | 2.1.0 |
| petsc | 3.14.1 | 3.14.1 | 3.14.1 |
| slepc | 3.14.0 | 3.14.0 | 3.14.0 |
| sundials | 3.1.2 | 3.1.2 | 3.1.2 |
| cantera | 2.4.0 | 2.4.0 | 2.4.0 |

Table 3: Softwares - Stacks versão 12

| Software | Docker | Singularity | Lmod |
|---|---|---|---|
| OS | Ubuntu 22.04 [imagem]; Windows [hospedeiro] | Ubuntu 22.04 [imagem]; Linux [hospedeiro] | Ubuntu 20.04, 22.04; Mint 21.1 |
| compilador | gnu 12.2 | gnu 12.2 | gnu 12.2 |
| python | 3.11.2 | 3.11.2 | 3.11.2 |
| valgrind | 3.20.0 | 3.20.0 | 3.20.0 |
| cmake | 3.25.2 | 3.25.2 | 3.25.2 |
| lapack | 3.11 | 3.11 | 3.11 |
| swig | 4.1.1 | 4.1.1 | 4.1.1 |
| boost | 1.81.0 | 1.81.0 | 1.81.0 |
| eigen3 | 3.4.0 | 3.4.0 | 3.4.0 |
| fmt | 9.1.0 | 9.1.0 | 9.1.0 |
| scons | 4.5.2 | 4.5.2 | 4.5.2 |
| googletest | 1.13.0 | 1.13.0 | 1.13.0 |
| coolprop | 6.5.0 | 6.5.0 | 6.5.0 |
| suitesparse | 7.0.1 | 7.0.1 | 7.0.1 |
| openmpi | 4.1.5 | 4.1.5 | 4.1.5 |
| gsl | 2.7.1 | 2.7.1 | 2.7.1 |
| hdf5 | 1.14.0 | 1.14.0 | 1.14.0 |
| trilinos/zoltan | 14.0.0 | 14.0.0 | 14.0.0 |
| scalapack | 2.2.0 | 2.2.0 | 2.2.0 |
| petsc | 3.19.0 | 3.19.0 | 3.19.0 |
| slepc | 3.19.0 | 3.19.0 | 3.19.0 |
| sundials | 6.5.0 | 6.5.0 | 6.5.0 |
| cantera | 2.6.0 | 2.6.0 | 2.6.0 |

## 1.3 Macros

Algumas macros **de exemplificação** são utilizadas nesse manual. A seguir, apresentamos quais e o que significam. Quando estiver lendo as próximas seções, se tiver alguma dúvidam, volte nesse tópico aqui para consultar o que a macro significa.

- **PATH_DO_FONTE_MFSIM**: local no seu sistema de arquivos onde o código fonte está;
- **PATH_DO_MFSIM_INSTALADO**: local no seu sistema de arquivos onde o MFSim foi instalado;

# 2 Configurando os ambientes

A configuração do ambiente é a instalação e configuração da *stack* sob a qual o MFSim executa e depende para funcionar. A *stack* varia de acordo com o sistema operacional e as ferramentas utilizadas. Algumas, como a lmod, são mais próximas das *stacks* configuradas nos clusters do MFLab (e outros clusters também). Outras, como o singularity e docker, estão mais próximas das *stacks* utilizadas em ambientes fora do MFLab. Como falado na seção anterior, escolha a stack que considerar mais viável para o seu uso.

## 2.1 Windows

### 2.1.1 Docker

O *docker* é uma plataforma que facilita o deployment de aplicações, permitindo através de seu sistema de imagens e containers, uma miríade de configurações de ambiente rodarem numa mesma máquina. Em termos simples, é uma forma de virtualização bem leve. Você pode aprender mais sobre como o docker funciona pelo site de seu desenvoledor, `https://docs.docker.com/get-started/overview/`.

Com essa ferramenta é possível rodar no Windows o MFSim, um software feito para executar em ambiente GNU/Linux. Entretanto, alguns pré-requisitos precisam ser observados **antes de tentar rodar o MFSim no Windows**.

**Primeiro** o docker precisa estar devidamente instalado e configurado na sua máquina. A instalação e os pré-requisitos do docker no Windows podem ser encontrados em `https://docs.docker.com/docker-for-windows/install/`

**Segundo** a ferramenta PowerShell, na versão 6 ou superior, deve estar instalada, configurada e habilitada para o usuário. Mais detalhes sobre isso, podem ser encontrados em `https://docs.microsoft.com/pt-br/powershell/scripting/install/installing-powershell-core-on-windows?view=powershell-7`

**Terceiro** o Docker cria um repositório local na máquina. O local varia, então será necessário privilégio de administrador na instalação do Docker, mesmo que temporariamente. Após a instalação, o local do repositório deve ter permissão de escrita para o usuário e o docker.

**Quarto** é necessário criar dois pontos no sistema de arquivos, com os quais o Docker será integrado. O usuário **deve** ter permissão de escrita nesses pontos. O primeiro ponto é para o código do MFSim e o segundo para o MFSim compilado. A sugestão é criar uma pasta na Área de Trabalho do usuário, para conter os dois pontos.

**Quinto** você precisará do código fonte do MFSim. Caso ainda não tenha, verifique a seção 1.1 para mais detalhes.

**Sexto** , acesso à internet, com o docker hub registry e o domínio mflab.mecanica.ufu.br (ambos HTTP/HTTPS), **sem quaisquer formas de bloqueios ou impedimento de acesso**.

**Sétimo** e último requisito é pelo menos 35 GB de espaço disponível em disco.

Uma vez satisfeitos **todos os requisitos**, você pode proceder com a instalação da imagem do mflab do Docker no Windows.

Para **fins de exemplificação**, vou considerar que o código fonte do MFSim está em `C:\Users\mflab\Desktop\MFSim\MFSim-cmake`

Abra o powershell e vá para a pasta *stacks/install_docker*, dentro do código fonte do MFSim e construa a imagem docker:

```
[PowerShell] > cd C:\Users\mflab\Desktop\MFSim\MFSim-cmake\stacks\install_docker\
C:\Users\mflab\Desktop\MFSim\stacks\install_docker\> docker build -t mflab_image .
```

O nome da imagem será `mflab_image` e será armazenada no repositório local do *docker*. Durante a construção da imagem será **necessário acesso à internet, em especial ao docker hub registry**.

Pode acontecer de um *warning* ser emitido ao final da construção da imagem, com uma mensagem parecida com a seguinte:

```
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host.
All files and directories added to build context will have '-rwxr-xr-x' permissions.
It is recommended to double check and reset permissions for sensitive files and directories.
```

Esse aviso decorre do fato do Windows não suportar permissão de execução, como nos sistemas unix-like (Linux, FreeBSD, etc). Essa situação não influencia no uso do *docker* e nem do MFSim, então pode ignorar.

Se nenhum outro erro ou aviso foi emitido, é porque a instalação da imagem deu certo e ela já está pronta para ser usada.

## 2.2 Linux

### 2.2.1 Singularity

O *singularity* é outra plataforma, que como o *docker*, facilita o *deploy* de aplicações. A diferença dele com o *docker* está no ambiente.

Enquanto o *docker* é de uso genérico, sendo muito utilizado para fazer deployment de aplicações web, o *singularity* é mais voltado a ambientes de alta performance como os encontrados em cluster.

A forma de uso, entretanto, é bem parecida. Você cria uma imagem com o seu ambiente/stack configurado e usa essa imagem para rodar o seu código em um *container*. Porém, existe uma diferença crucial. Enquanto o *docker* simula uma máquina virtual (muito leve), com ambiente e sistema de arquivos próprios dentro de um container praticamente isolado da máquina local, o *singularity* usa o ambiente e o sistema de arquivos da máquina local no *container*, funcionando como uma camada entre o seu código e o sistema local. Dessa maneira, você consegue efetivamente interferir e controlar, de forma simples, o que está acontecendo dentro do container.

Isso é particularmente útil em clusters, pois permite o uso do *container* com softwares de execução em lotes, como o *pbs*, *torque* ou *slurm*, que requerem controle direto sobre o que a aplicação dentro do container está fazendo, como por exemplo, quantos processos filhos ela abre, quanto de RAM ela consome, entre outras coisas.

Por conta disso, se você for rodar o MFSim em um cluster fora do MFLab, é altamente provável que o rodará via *singularity*.

Uma última observação é que não precisa criar ponto específico no seu sistema de arquivos para o *singularity*. Como ele se integra com o sistema hospedeiro, podendo dentro do *container* navegar a vontade pelo sistema de arquivos local, não há necessidade de criar um ponto específico onde o *singularity* irá se conectar, como acontece com o *docker*.

Agora vamos à instalação dessa *stack*.

Assim como no *docker*, **você precisa primeiro instalar o *singularity*** na sua máquina, **antes** de fazer qualquer outra coisa proposta nesse tutorial.

Neste sentido você pode tanto baixar e instalar o *singularity* diretamente do site do fabricante (`https://sylabs.io/guides/3.7/admin-guide/installation.html`) ou pode utilizar o instalador do *singularity*, fornecido junto ao código do MFSim, na pasta *stacks*. Aqui, vamos demonstar como instalar via instalador já que para o caso de instalação manual, as instruções estão no site do fabricante.

#### 2.2.1.1 Instalando o singularity via instalador

Abra o terminal e execute:

```
$ cd PATH_DO_FONTE_MFSIM/stacks/install_singularity
$ sudo ./install.sh home=$HOME
```

Isso fará a instalação do go, singularity e das imagens na sua máquina. Após a instalação, pode ser necessário abrir outro terminal ou até mesmo realizar um novo login no sistema a fim de atualizar as variáveis de ambiente. Feito isso, abra um terminal e verifique se o *singularity* está de fato instalado:

```
$ singularity
Usage:
  singularity [global options...] <command>
```

Se o resultado for parecido com o mostrado acima, então o *singularity* está instalado corretamente.

O instalador já baixa e configura as imagens, então se a instalação foi bem sucedida, você já tem todo o necessário para usar o *singularity* na sua máquina.

#### 2.2.1.2 Imagens e versões da stack

Há 3 imagens disponíveis, 1 para a stack singularity na versão 10 e 2 para a versão 12. O instalador criará macros para cada uma, que são

Table 4: Macros das imagens Singularity

| Versão | Macro |
|---|---|
| 10 | `MFLAB_IMAGE_GCC10` |
| 12 | `MFLAB_IMAGE_GCC12_DEBUG` |
| | `MFLAB_IMAGE_GCC12_ALEX` |

Há duas versões para stack singularity versão 12. Uma, contendo o cantera 2.6.0 original, a `MFLAB_IMAGE_GCC12_DEBUG` e outra, contendo o cantera 2.6.0 modificado pelo Alex, a `MFLAB_IMAGE_GCC12_ALEX`. Ao usar a versão 12, exceto se você for precisar do cantera modificado pelo Alex, você deve usar a imagem `MFLAB_IMAGE_GCC12_DEBUG`.

#### 2.2.1.3 Testando a instalação

Por fim você pode testar a instalação tentanto executar a imagem via *singularity*:

```
$ singularity shell $MFLAB_IMAGE_GCC12_DEBUG
Singularity>
```

Se o seu resultado for parecido com o acima, então o *singularity* está pronto para ser usado com o MFSim.

### 2.2.2 Lmod

Todo sistema operacional utiliza de certas variáveis para saber quais softwares estão instalados nele, onde estão os arquivos de desenvolvimento, onde estão as bibliotecas, entre outras coisas. A essas variáveis chamamos de **variáveis de ambiente**. Isso vale para **praticamente todos os sistema operacionais**, ou seja, Windows, Linux, Mac OS X, Android, iOS, Solaris, todos tem em comum o uso de variáveis de ambiente (embora o nome deles varie de acordo com o sistema operacional).

O Lmod é um software que gerencia variáveis de ambiente, utilizando um sistema de módulos. Com ele você pode instalar um conjunto de softwares, onde desejar, da forma que desejar e integrado com quem desejar, e adicionar as configurações desses softwares às variáveis de ambiente do sistema operacional, de forma que ele identifique os softwares que você instalou como parte do sistema.

Isso é extremamente útil quando você precisa ter vários tipos de softwares e bibliotecas instalados ao mesmo tempo em seu computador, incluindo quando esses softwares não são compatíveis entre si. Ex: você não pode ter dois compiladores GCC ao mesmo tempo na máquina. Um deles precisará ter um nome diferente para não conflitar com o outro. O lmod resolve esse problema.

Por este motivo o lmod provê a *stack* mais sofisticada, visto que permite grande flexibilidade ao usuário, porém é também a mais trabalhosa de instalar. Os clusters do MFLab utilizam essa *stack*, então usar lmod na sua máquina pode facilitar e muito na hora de utilizar os clusters depois.

Além disso, o lmod, devido a sua proposta de funcionamento, termina por não "amarrar" as bibliotecas e softwares instalados nele ao sistema operacional, ficando o usuário livre para atualizar os pacotes como precisar, assim como instalar múltiplas versões das bibliotecas, da mesma forma que acontece nos clusters.

O único cuidado que precisará ter é com a versão do compilador. Como tudo é construído com base nela, se mudar a versão do compilador, especialmente uma versão major (mudança de gcc 9 para gcc 10 por exemplo), precisará então recompilar a stack inteira.

Fora isso, poderá mudar o que quiser, da forma que quiser.

Uma última observação a se fazer é que essa sequência de instalação foi **testada no Ubuntu 20.04 e Mint 21.4**. Se está utilizando outra distribuição, pode ser que será necessário adaptar alguns comandos, como os pacotes a serem instalados do repositório da distribuição ou alguns paths de inclusão e linkagem nos módulos.

Exemplificando o caso acima, em sistemas Debian based (Debian, Ubuntu, Mint, etc), é comum os arquivos de biblioteca (os **.a** e **.so**) serem instaladas num diretório chamado **lib** a partir do path de instalação da biblioteca. Já em sistemas Red-Hat based (Red Hat, CentOS, SuSE, OpenSuSE, Fedora, etc), os mesmos arquivos são comumente instalados no diretório **lib64**. Logo, se estiver usando um sistema Red-Hat based, precisará alterar as variáveis `LD_LIBRARY_PATH` e `LIBRARY_PATH` dos módulos de `$prefix/lib` para `$prefix/lib64`.

#### 2.2.2.1 Instalando a lmod

O MFLab provê um instalador para a *stack* lmod, dentro da pasta *stacks* do MFSim, desenvolvido em python 3. Por conseguinte, você precisará do python 3 instalado na sua máquina **previamente** a instalação da lmod. Também precisará de acesso à internet, especialmente ao site do MFLab, `https://www.mflab.mecanica.ufu.br`, pois o instalador fará o download de arquivos de configuração do site do MFLab durante a instalação.

Também verifique se possui permissão de sudo:

```
$ sudo ls /
```

Ao executar o comando acima será pedido uma senha, que é a mesma que usa para o logar o seu usuário na máquina. Se você não setou senha para o seu usuário (faz autologin), sete, pois precisará dela para continuar a instalação. Também, se ao fornecer sua senha de usuário o comando acima falhar, é porque você não tem permissões de sudo. Precisará ajustar isso **antes** de prosseguir.

Uma vez que possua todos os requisitos, você pode instalar a *stack* lmod na sua máquina digitando no terminal:

```
$ cd PATH_DO_FONTE_MFSIM/stacks/install_lmod
$ ./install.py install_path=/opt/ohpc install_lmod=true sudo_pwd="MINHA SENHA DE SUDO"
```

O instalador é bem versátil e pode ser utilizado para além de instalar a *stack* lmod, a atualizar com novas versões de bibliotecas. Você pode conferir quais são as opções digitando no terminal:

```
$ ./install.py
```

Assim que a instalação terminar, **será necessário abrir um novo terminal ou até mesmo fazer um novo login**, a fim de atualizar as variáveis de ambiente.

Para testar a instalação, acesse o terminal e digite:

```
$ ml
No modules loaded
```

Se o seu resultado for parecido com o mostrado acima, então a *stack* lmod está funcionando e já pode ser utilizada com o MFSim.

#### 2.2.2.2 Atualizando a lmod

Caso você já possua a *stack* lmod instalada e deseja apenas atualiza-la, instalando uma stack nova configurada após já ter a lmod instalada na sua máquina, você pode se utilizar do instalador da seguinte forma:

```
$ cd PATH_DO_FONTE_MFSIM/stacks/install_lmod
$ ./install.py install_path=/opt/ohpc sudo_pwd="MINHA SENHA DE SUDO"
```

## 2.3 Gmsh

Dos programas utilizados para montar as simulações no MFSim, o único não contemplado em qualquer *stack* é o gerador de malhas *gmsh*. Porém, a instalação do mesmo bem é simples e pode ser realizada através do terminal, **APÓS A INSTALAÇÃO DA STACK** escolhida. Para versões do Linux baseadas em pacotes *.deb* (Linux Ubuntu, Linux Debian, Linux Mint e seus derivados), a instalação pode ser realizada utilizando o comando:

```
sudo apt install gmsh
```

Após a instalação o *gmsh* pode ser inicializado via terminal com o comando

```
gmsh &
```

onde o & coloca a execução do programa em background, liberando a janela do terminal utilizada para outras operações.

Para usuário Windows, você pode baixar o *gmsh* para Windows diretamente do site do seu desenvolvedor, `https://gmsh.info/`.

# 3 Filosofia de uso

Agora que instalou o ambiente para utilizar o MFSim, vamos entender a filosofia de uso do mesmo.

**Atenção!** A ideia dessa seção é explicar alguns detalhes sobre como se usa o MFSim. Logo, os **comandos nessa seção são apenas para mostrar a sequência de uso e não para exemplificar comandos reais em qualquer situação**.

Ao final dessa seção está uma subseção com detalhes de como aplicar os comandos para cada uma das *stacks* já descritas nesse manual. Entretanto, recomendamos que não tente simplesmente copiar e colar os comandos. Entenda a sequência apresentada e como eles se relacionam com cada stack. O "Manual de Usuário" apresenta de forma mais detalhada como aplicar os comandos, configurar o MFSim, entre outras informações pertinentes ao uso diretamente.

## 3.1 Pastas de compilação e Instalação

O MFSim é fornecido via códifo fonte, o que implica que precisará compilar, gerar o programa executável do MFSim, a partir do seu código fonte. Isso é feito através das ferramentas de compilação e bibliotecas embutidas na *stack* que instalou na seção anterior. Após compilar o MFSim, você irá executá-lo.

A primeira etapa é a de **compilação**. Nela utilizaremos o `cmake` para gerar as instruções de compilação e em seguida executar a compilação propriamente dita. O `cmake` requer o uso de uma pasta de compilação **separada** da pasta do código fonte. Então, após ter realizado o download do MFSim, precisará criar uma pasta para o `cmake`. Para fins de exemplificação, chamaremos ela de **build** e a criaremos dentro da pasta com o código fonte do MFSim.

```
PATH_DO_FONTE_MFSIM
        /bin
        /build <- crie esta pasta aqui!
        /cmake
        /CMakeLists.txt
        /config.h.in
        /docs
        /geo
        /input
        /libs
        /parser
        /README.md
        /src_amr
        /src_check
        /src_dl
        /src_lag
        /src_log
        /src_mat
        /src_par
        /src_term
        /src_turb
        /src_vof
        /stacks
        /tests
        /tmp
        /tools
```

Uma vez dentro da pasta **build**, executaremos o `cmake`. Há alguns detalhes nessa parte dependendo da *stack* que está utilizando. Por isso, as sequências de comandos mostradas a seguir são explicativas, ou seja, **não tente executar os comandos imediamente**. A ideia aqui é entender o que eles fazem. Detalhes da execução, conforme a *stack* estão nas próximas seções desse manual. Entendido esse ponto, vamos aos comandos.

```
$ cd PATH_DO_FONTE_MFSIM/build
$ cmake .. -Wno-dev
```

Se nenhum erro ocorrer, o `cmake` terá gerado a receita de compilação, o arquivo `Makefile` que utilizaremos para compilar o MFSim. Isto faremos executando o comando `make`, dendo da pasta `build`.

```
$ make
```

Novamente, se tudo ocorreu sem erros, a próxima etapa é **instalar** o MFSim. Faremos isso utilizando o comando:

```
$ make install
```

Após esse comando, teremos o MFSim instalado e pronto para uso.

Antes de irmos para as próximas seções que mostram como aplicar os comandos mostrados aqui para cada uma das *stacks*, vamos revisar a ideia desse bloco aqui, que mostra a **filosofia de uso do MFSim**.

- 1 - O MFSim é fornecido via código fonte, então precisa gerar a versão executável e instalar no seu sistema;
- 2 - A etapa de gerar o executável chamamos de compilação;
- 3 - A compilação segue a sequência: cria a pasta `build` -¿ executa o `cmake` -¿ executa o `make`
- 4 - A instalação é feita executando o `make install` **após** a compilação terminar
- 5 - O MFSim é utilizado onde foi instalado

## 3.2 Escolhendo o local de instalação

É possível definir um local no seu sistema onde você quer instalar o MFSim. Por padrão, o `cmake` tentará instalar em um local pré-definido, dependendo da *stack* utilizada.

### 3.2.1 Lmod e Singularity

Nas *stacks* lmod e Singularity o `cmake` tenta instalar o MFSim na pasta `install`, dentro do código fonte do MFSim.

```
PATH_DO_FONTE_MFSIM
        /bin
        /build
        /cmake
        /CMakeLists.txt
        /config.h.in
        /docs
        /geo
        /input
        /install <- cmake cria e tenta instalar o MFSim aqui
        /libs
        /parser
        /README.md
        /src_amr
        /src_check
        /src_dl
        /src_lag
        /src_log
        /src_mat
        /src_par
        /src_term
        /src_turb
        /src_vof
        /stacks
        /tests
        /tmp
        /tools
```

Esse é o comportamento padrão, porém você pode especificar um outro local para a instalação. Para isso, ao executar o `cmake` na etapa de compilação, utilize o opção `-Dprefix` para indicar o local. Ex:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ cmake .. -Dprefix=/tmp/mfsim -Wno-dev
```

A sequência acima configurará o `cmake` para instalar o MFSim, **após** a compilação, na pasta `/tmp/mfsim`.

Em geral, você pode especificar qualquer local para a instalação do MFSim, **desde que possua permissões de escrita nesse local**.

### 3.2.2 Docker

O Docker já traz o local de instalação do MFSim configurado por padrão e você **não precisa e nem deve alterar isso**. Nessa *stack* o MFSim sempre é instalado na pasta `/mfsim`

## 3.3 Modos de Instalação

O modo padrão de instalação do MFSim é o *debug*, que habilita funções de debugging como tabela de símbolos úteis ao gbd e valgrind. Entretanto, é possível utilizar a versão *release*, que desabilita as funções de debug e otimiza o código, tornando a execução mais rápida.

Qual das duas versões instalar dependerá do seu uso do MFSim. Se está utilizando o MFSim em ambiente de desenvolvimento é **fortemente recomendado** utilizar o modo padrão, o *debug*.

Já se estiver utilizando o MFSim em ambiente de produção, então é recomendado utilizar a versão otimizada, a *release*.

Para ativar o modo *release* simplesmente fornece o parâmetro de modo ao cmake:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ cmake .. -Drelease=1
```

## 3.4 Instalação de testes

O MFSim também provê testes de verificação do mesmo, porém eles não são instalados por padrão. Para habilitar a instalação dos testes use:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ cmake .. -Dinstall_tests=1
```

Mais detalhes de como utilizar os testes estão na seção 6

# 4 Compilando e Instalando

Essa seção mostra como compilar e instalar o MFSim, por *stack*. Você não precisa ler toda essa seção e pode ir direto para a subseção da *stack* que estiver utilizando.

## 4.1 Lmod

Conforme explicado na seção 2.2.2, o lmod é um software de gestão de variáveis de ambiente configurado via módulos. Então, se está utilizando lmod, precisará carregar os módulos com as bibliotecas que o MFSim precisa.

Basicamente o MFSim precisa de:

1. compilador C/C++ e fortran, também necessário para definir a versão usada;
2. cmake (apenas para compilação);
3. uma biblioteca MPI;
4. HDF5;
5. Cantera, GSL, zoltan, slepc e outras libs;

Então, carregaremos os módulos que proveem esses requisitos, teremos **algo como**:

```
$ ml gnu/VERSÃO cmake openmpi hdf5 cantera gsl slepc zoltan
```

### 4.1.1 Módulos versão 10

Aplicando o comando de exemplo anterior a versão 10 da *stack* lmod teremos:

```
$ ml gnu/10.2.0 cmake openmpi hdf5 cantera gsl slepc zoltan
```

### 4.1.2 Módulos versão 12

Aplicando o comando de exemplo anterior a versão 12 da *stack* lmod teremos:

```
$ ml gnu/12.2.0 cmake openmpi hdf5 cantera gsl slepc zoltan coolprop
```

### 4.1.3 Compilando e instalando

Agora que temos os módulos carregados, podemos continuar com a compilação e instalação:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ cmake .. -Wno-dev
$ make
$ make install
```

Se nenhum erro for acusado até aqui, o MFSim foi compilado e instalado com sucesso.

Detalhes de como usar o MFSim estão no manual do usuário no site oficial do MFSim: `https://www.mflab.mecanica.ufu.br/mfsim/`.

Por fim, recomendo que termine a leitura do manual, especialmente a seção 6 que contém toda uma sequência de passos para verificar se a instalação do MFSim está de fato correta.

## 4.2 Singularity

Assumindo que seguiu as instruções de instalação e configuração do Singularity conforme descritas na seção 2.2.1, para compilar e instalar o MFSim, precisará então instanciar o singularity e seguir a sequência de compilação e instalação, **dentro do singularity**.

Rememorando, para instanciar o singularity você utiliza algo como:

```
$ singularity shell $MFLAB_IMAGE_GCC12_DEBUG
Singularity>
```

Sendo que esse `Singularity>` indica que você está dentro do singularity. Continuando,

```
Singularity> PATH_DO_FONTE_MFSIM/build
Singularity> cmake .. -Wno-dev
Singularity> make
Singularity> make install
```

Se nenhum erro for acusado até aqui, o MFSim foi compilado e instalado com sucesso.

Detalhes de como usar o MFSim estão no manual do usuário no site oficial do MFSim: `https://www.mflab.mecanica.ufu.br/mfsim/`.

Por fim, recomendo que termine a leitura do manual, especialmente a seção 6 que contém toda uma sequência de passos para verificar se a instalação do MFSim está de fato correta.

## 4.3 Docker

Por fim, temos a *stack* Docker apresentada na seção 2.1.1. O docker utiliza volumes virtuais para conectar o *container* ao sistema de arquivos da máquina hospedeira. Nesse sentido, dois volumes estão configurados na imagem: um para a compilação e outro para a execução do MFSim.

Logo, ao instanciarmos o docker, precisaremos configurar esses dois volumes. O comando a seguir apresenta um **exemplo** disso:

```
docker run -it -v PATH_DO_FONTE_MFSIM:/mflab -v PATH_DO_MFSIM_INSTALADO:/mfsim mflab_image /bin/bash
```

Como pode ver, dentro do *container* docker o `/mflab` conterá o código fonte do MFSim, onde você fará a compilação, enquanto que o `/mfsim` conterá o MFSim instalado, de onde você executará suas simulações.

Uma vez que tenha instanciado o docker, poderá então seguir com a sequência de compilação e execução:

```
[bash] # cd /mlab/build
[bash] # cmake .. -Wno-dev
[bash] # make
[bash] # make install
```

Se nenhum erro for acusado até aqui, o MFSim foi compilado e instalado com sucesso.

Detalhes de como usar o MFSim estão no manual do usuário no site oficial do MFSim: `https://www.mflab.mecanica.ufu.br/mfsim/`.

Por fim, recomendo que termine a leitura do manual, especialmente a seção 6 que contém toda uma sequência de passos para verificar se a instalação do MFSim está de fato correta.

# 5 Ferramentas de Apoio

Junto ao MFSim estão algumas ferramentas que fornecem funções adicionais. Essa seção descreve estas ferramentas.

## 5.1 Ferramenta de Limpeza

Durante a compilação do MFSim é comum o compilador gerar arquivos de módulos fortran, os arquivos com a extenção `.mod` dentro dos diretórios onde está o código fonte e depois os mover para o diretório de compilação. Se por algum motivo houver uma falha de compilação, pode acontecer desses arquivos `.mod` ficarem "soltos" nas pastas de código fonte e atrapalharem uma compilação posterior.

Pra evitar essa situação, temos uma ferramenta que acessa todas as pastas e subpastas onde há código fonte e remove os arquivos `.mod` "soltos".

Para usar essa ferramenta, vá para a pasta onde instalou o MFSim:

```
$ cd PATH_DO_MFSIM_INSTALADO/bin
$ ./clear_mod
```

## 5.2 Ferramenta de Teste de Casos

O MFSim é um software CFD o que implica que sua função primária é simular o comportamento de fluídos dinâmicos utilizando técnicas computacionais. Logo, para saber se o MFSim está funcionando corretamente é necessário executar alguma simulação nele.

Nesse sentido esta ferramenta verifica se o código fonte está correto, diante de algumas simulações de teste do sistema.

Isso serve a dois propósitos: permitir ao desenvolvedor verificar se suas alterações no código fonte não danificaram o fluxo de execução do sistema e permitir ao usuário verificar se o sistema está ou não correto.

Atualmente os seguintes casos de teste estão implementados no sistema:

- Solução manufaturada: o procedimento básico deste caso envolve a definição de uma solução manufaturada para cada variável do problema, de forma que cada variável está presente nas equações diferenciais do escoamento;
- Escoamento sobre uma esfera estacionária: esse caso é um escoamento monofásico sobre uma esfera estacionária;

A ferramenta configura o MFSim da forma necessária para cada caso, executa a simulação e a valida os resultados. É possível executar cada teste separadamente ou executar todos os testes disponíveis. Caso a execução ocorra sem problemas, ao final a ferramenta informa isso ao usuário. Já caso algum erro seja encontrado, a ferramenta aponta onde tal erro aconteceu.

Dito isso, vejamos como usar a ferramenta.

```
$ cd PATH_DO_MFSIM_INSTALADO/bin
$ ./run_tests [-path PATH_PARA_EXECUÇÃO_DO_TESTE] [-test NOME_DO_TESTE]
```

É possível especificar um local específico no seu sistema de arquivos onde os testes serão executados. **Isso não é obrigatório** e caso esteja utilizando a *stack* Docker não é recomendado. Caso esteja utilizando lmod ou singularity e deseje especificar o local, utilize a opção `-path`.

Da mesma maneira, é possível especifical qual teste deseja rodar. O comportamento padrão é rodar todos os testes, porém você pode rodar apenas um. Se esse for o caso, utilize a opção `-test` fornecendo o nome do teste que deseja executar. Essa opção pode ser utilizada em qualquer uma das stacks.

Os testes disponíveis para `NOME_DO_TESTE` são:

- **manuf**: para a solução manufaturada;
- **sphere**: para o escoamento sobre uma esfera estacionária;

Sendo assim, caso deseje rodar o teste da solução manufaturada:

```
$ cd PATH_DO_MFSIM_INSTALADO/bin
$ ./run_tests -test manuf
```

E da esfera:

```
$ cd PATH_DO_MFSIM_INSTALADO/bin
$ ./run_tests -test sphere
```

É possível ainda não informar qualquer parâmetro e executar todos os testes disponíveis, **sequencialmente**:

```
$ cd PATH_DO_MFSIM_INSTALADO/bin
$ ./run_tests
```

# 6 Homologação

Esta seção trata sobre a homologação do MFSim. Como é possível executar o mesmo em vários sistemas operacionais e configurações de ambiente, a seção está organizada por sistema operacional e ambiente, ou *stack* utilizada.

Por este motivo, você só precisa seguir os passos de homologação da *stack* que estiver utilizando, podendo ignorar as demais.

O processo de homologação é razoavelmente simples e consiste na *checklist*:

- 1 - Escolher um sistema operacional e ambiente no qual deseja rodar o MFSim;
- 2 - Instalar e configurar as bibliotecas e dependências do MFSim;
- 3 - Baixar e compilar o código do MFSim
- 4 - Validar o MFSim

Nas próximas seções, estão as *checklists* por sistema operacional e *stack*

## 6.1 Windows

Nesse sistema operacional temos apenas 1 *stack*, para a qual veremos a seguir como homologar.

### 6.1.1 Docker

#### 6.1.1.1 Instalação das bibliotecas e dependências

Instale e configure o Docker conforme mostrado na seção 2.1.1

#### 6.1.1.2 Obtendo o MFSim e configurando o Docker para uso

Faça o download do código fonte do MFSim caso já não o tenha feito. Descompacte o arquivo baixado em algum local do seu sistema de arquivos, o que gerará uma pasta, que será a nossa pasta de código fonte. Dentro dessa pasta, crie outra para ser a nossa pasta de compilação. Por convenção, vamos chamá-la de `build`. Crie uma outra pasta no seu sistema de arquivos para ser a nossa pasta de execução.

**Para fins de exemplificação**, vou considerar que as pastas são

```
C:\Users\mflab\Desktop\MFSIM\src    <- a pasta com o código fonte
C:\Users\mflab\Desktop\MFSIM\install  <- a pasta que usaremos para execução
```

Agora vamos instanciar o Docker em modo interativo, via PowerShell, apontando o volume virtual `/mflab` para a pasta gerada durante a descompactação do código fonte e o volume virtual `/mfsim` para a pasta de execução:

```
[PowerShell] > docker run -it -v C:\Users\mflab\Desktop\MFSIM\src:/mflab -v
↪  C:\Users\mflab\Desktop\MFSIM\install:/mfsim mflab_image /bin/bash
```

#### 6.1.1.3 Compilando o MFSim

Uma vez instanciado o container Docker, vamos compilar e instalar o MFSim.

```
[bash] # cd /mflab/build
[bash] # cmake .. -Dinstall_tests=1 -Wno-dev
[bash] # make
[bash] # make install
```

Se nenhum erro aconteceu durante a compilação, então pode proceder para a próximo ponto. Caso contrário, você pode reportar o problema que encontrou para `suportemflab@mecanica.ufu.br` a fim de obter uma solução para o mesmo.

#### 6.1.1.4 Validando

Para finalizar a homologação, vamos validar o MFSim. Para isso faremos uso da ferramenta de testes, já compilada na etapa anterior que realiza a validação automática:

```
[bash] # cd /mfsim/bin
[bash] # ./run_tests
```

A validação poderá rodar por algumas horas, dependendo da configuração e hardware da sua máquina. Se ao final da execução da ferramenta aparecer a mensagem

```
************* MFSim seems to be correct! *************
```

Então o código do MFSim está válido e homologado.

## 6.2 Linux

Nesse sistema operacional temos 3 *stacks* diferentes, conforme escrito na seção 1. A seguir, veremos como homologar cada uma dessas *stacks*.

### 6.2.1 Singularity

#### 6.2.1.1 Instalação das bibliotecas e dependências

Instale e configure o Singularity conforme mostrado na seção 2.2.1

#### 6.2.1.2 Obtendo o MFSim e configurando-o para uso

Faça o download do código fonte do MFSim caso já não o tenha feito. Descompacte o arquivo baixado em algum local do seu sistema de arquivos. A esse local, para fins de exemplificação e demonstração de comandos, chamaremos de `PATH_DO_FONTE_MFSIM`.

Dentro do `PATH_DO_FONTE_MFSIM` crie uma subpasta de compilação, que para fins de exemplificação, chamaremos aqui essa subspasta de `build`

Por fim, instancie o Singularity em modo interativo dentro da pasta de compilação:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ singularity shell $MFLAB_IMAGE
```

#### 6.2.1.3 Compilando o MFSim

Uma vez instanciado o Singularity, vamos compilar e instalar o MFSim.

```
Singularity> cmake .. -Dinstall_tests=1 -Wno-dev
Singularity> make
Singularity> make install
```

Ao final dessa sequência, o MFSim estará instalado na pasta `PATH_DO_FONTE_MFSIM/install`, que chamaremos de `PATH_DO_MFSIM_INSTALADO`.

Se nenhum erro aconteceu durante a compilação, então pode proceder para a próximo ponto. Caso contrário, você pode reportar o problema que encontrou para `suportemflab@mecanica.ufu.br` a fim de obter uma solução para o mesmo.

**Ajuda:** Caso não esteja se lembrando o que são as macros `PATH_DO_FONTE_MFSIM` e `PATH_DO_MFSIM_INSTALADO`, dê uma olhada nas seções 1.3 e 4.2.

#### 6.2.1.4 Validando

Para finalizar a homologação, vamos validar o MFSim. Para isso faremos uso de outra ferramenta, já compilada na etapa anterior que realiza a validação automática:

```
Singularity> cd PATH_DO_MFSIM_INSTALADO/bin
Singularity> ./run_tests
```

A validação poderá rodar por algumas horas, dependendo da configuração e hardware da sua máquina. Se ao final da execução da ferramenta aparecer a mensagem

```
************* MFSim seems to be correct! *************
```

Então o código do MFSim está válido e homologado.

### 6.2.2 Lmod

#### 6.2.2.1 Instalação das bibliotecas e dependências

Instale e configure o Lmod conforme mostrado na seção 2.2.2

#### 6.2.2.2 Obtendo o MFSim e configurando-o para uso

Faça o download do código fonte do MFSim caso já não o tenha feito. Descompacte o arquivo baixado em algum local do seu sistema de arquivos. A esse local, para fins de exemplificação e demonstração de comandos, chamaremos de `PATH_DO_FONTE_MFSIM`.

Dentro do `PATH_DO_FONTE_MFSIM` crie uma subpasta de compilação, que para fins de exemplificação, chamaremos aqui essa subspasta de `build`

#### 6.2.2.3 Compilando o MFSim

Instancie os módulos lmod necessários ao MFSim, **conforme a versão da lmod que estiver utilizando**: compilador, `MPI` e `HDF5 e outras libs MPI`.

Versão 10:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ ml gnu/10.2.0 cmake openmpi hdf5 cantera gsl slepc zoltan
```

Versão 12:

```
$ cd PATH_DO_FONTE_MFSIM/build
$ ml gnu/12.2.0 cmake openmpi hdf5 cantera gsl slepc zoltan coolprop
```

Uma vez que os módulos estiverem instanciados, compile e instale:

```
$ cmake .. -Dinstall_tests=1 -Wno-dev
$ make
$ make install
```

Ao final dessa sequência, o MFSim estará instalado na pasta `PATH_DO_FONTE_MFSIM/install`, que chamaremos de `PATH_DO_MFSIM_INSTALADO`.

Se nenhum erro aconteceu durante a compilação, então pode proceder para a próximo ponto. Caso contrário, você pode reportar o problema que encontrou para `suportemflab@mecanica.ufu.br` a fim de obter uma solução para o mesmo.

**Ajuda:** Caso não esteja se lembrando o que são as macros `PATH_DO_FONTE_MFSIM` e `PATH_DO_MFSIM_INSTALADO`, dê uma olhada nas seções 1.3 e 4.1.

#### 6.2.2.4 Validando

Para finalizar a homologação, vamos validar o MFSim. Para isso faremos uso da ferramenta de testes, já compilada na etapa anterior que realiza a validação automática:

```
$ cd PATH_DO_MFSIM_INSTALADO/bin
$ ./run_tests
```

A validação poderá rodar por algumas horas, dependendo da configuração e hardware da sua máquina. Se ao final da execução da ferramenta aparecer a mensagem

```
************* MFSim seems to be correct! *************
```

Então o código do MFSim está válido e homologado.