RootGen
Documentação Resumida

Autor: Felipe Heiji Takaoka

Observações preliminares

- **Para iniciar o programa, digite "RootGen" na linha de comando do Matlab;**
- **Este trabalho foi largamente baseado no artigo DUPUY, L., T. FOURCAUD, AND A. STOKES. 2005b. A numerical investigation into the influence of soil type and root architecture on tree anchorage. Plant and Soil 278: 119?134;**
- As documentações deste programa são extensas e, portanto, foram separadas em diferentes arquivos de forma a facilitar a leitura deles para uso e edição do programa;
- Os seguintes conhecimentos são aconselháveis para a total compreensão do software, em ordem de prioridade:
  - Programação orientada a objetos (métodos e atributos, herança, ponteiros, etc);
  - Estrutura de dados do tipo lista ligada e árvore, e seus algoritmos de percurso (recursivos);
  - Interfaces gráficas (GUIs);
  - Algebra linear (produto escalar, norma, ortogonalidade, matriz de rotação, ângulos de Euler);
  - Métodos numéricos (básico - critério de parada e método da bissecção);
  - Noções de probabilidade (básico - diferentes tipos de distribuição probabilística);
- Para definições e notações utilizadas, consulte o arquivo "**Notation**". Muitas das notações foram baseadas naquelas apresentadas no artigo de Godin e Garaglio e sua leitura é altamente recomendada;
- Para a visualização da organização geométrica e topológica das classes, consulte o arquivo "**Class representation**";
- Para a visão geral dos componentes da interface gráfica do RootGen, consulte o arquivo **"GUI"**;
- **RootGen** foi desenvolvido orientado a objetos no Matlab com o padrão de arquitetura MVC;
  - A pasta "**documentation**" concentra arquivos relacionados a documentação do programa;
  - A pasta "**model**" concentra as classes do modelo de dados do programa;
  - A pasta "**parameters**" concentra atributos dos parâmetros pré-definidos de padrões de raízes ou criados pelo usuário;
  - A pasta "**scripts**" concentra os scripts python gerados pelo RootGen;
  - A pasta "**view**" concentra as funções e interfaces gráficas (GUI) do programa;
- A programação orientada a objetos do Matlab possui algumas peculiaridades:
  - Chamada de métodos de instância de duas formas: *obj.method(args)* ou *method(obj, args)*;
  - Para poder fazer passagens por referência, uma classe deve herdar de ***handle (eq. ponteiro)***;
  - Para subclasses de uma mesma classe poderem ser armazenadas num mesmo array, a superclasse deve herdar de ***matlab.mixin.Heterogeneous***;
  - Atribuições para atributos com método setter utilizam a função automaticamente sem a necessidade de chamá-la explicitamente;
- O modelo do programa é baseado em **três escalas de representação** (modularidades MTG): root structure, axes e nodes;
- A representação computacional da estrutura radicular é feita a partir da estrutura de dados do tipo **árvore**/grafo. Desta forma, muitos dos métodos de percurso são recursivos e assim é recomendável que o leitor os conheça (*preorder, inorder, postorder*);
- A geração da raiz é realizada em **três etapas principais**:
  - **Inicialização**: Definição e inicialização dos parâmetros característicos da raiz;
  - **Crescimento:** Incremento passo a passo de segmentos de raiz discretizados de todos os axes ativos da estrutura até que todos tenham atingido uma condição de contorno;
  - **Preenchimento:** Atribuição de diâmetros a todos os segmentos de raiz de forma a obter um volume fixo final da raiz completa;
- As interfaces gráficas foram criadas usando o **GUIDE** do Matlab;
- **Para mais informações sobre as classes e seus métodos e atributos associados: "doc <*classe*>" ou "help <*classe/método/atributo*>". Utilize-os extensivamente! Muito trabalho**

foi empregado nessa documentação!!! É possível, contudo, devido a diversas modificações e atualizações, que alguns deles não estejam atualizados e que hajam bugs e defeitos no programa (vide to-do's). Desta forma, é importante também a compreensão do código de cada método.

- Para alterar os parâmetros da raiz a partir da interface gráfica (GUI), selecione a opção **"User-defined"** em "Root Structure Initialization" e em seguida vá ao menu em "**Options > Change Constants**";
- O programa não está implementado na forma mais ótima possível tanto do ponto de vista organizacional quanto do ponto de vista de custo computacional. Desta forma, tanto para entender melhor a construção do RootGen quanto para melhorá-lo, o leitor poderá realizar os to-do's (ideias de tarefas ainda não implementadas ou a modificar), que se encontram mais adiante neste documento.

Dicas para debugs e implementações

- Para debugar/explorar os dados da raiz criada a partir da interface gráfica (GUI): "**File > Save to workspace**" exporta o objeto **RootStruct r** para o workspace do Matlab;
- Para testar pequenas modificações, altere os parâmetros da raiz (pela GUI ou manualmente - ver documentação da classe *Parameters* e do método *saveParams()*) para facilitar seus testes e utilize um script (*drive.m*).
- Métodos úteis para testes (para mais informações sobre eles, consulte sua respectiva documentação):
  - *Parameters/Parameters('file','<seu arquivo de parametros salvo>.mat'): cria um objeto Parameters para criar uma RootStruct a partir das raizes padrões, salvas ou definidas numa struct;*
  - *RootStruct/stepGrow(): itera etapa por etapa dentro do processo de crescimento;*
  - *RootStruct/printRootStruct(): imprime na tela a raiz completa na sintaxe MTG;*
  - *RootStruct/drawRootStruct(): plota a raiz;*
  - *RootStruct/getNodeFromGlbIndex(): retorna o ponteiro/handle para um nó com o ID dado;*
  - *Axe/nodeFromIndex(): retorna o ponteiro/handle para o iésimo nó do eixo;*
  - *Axe.printNodeCoords(): imprime na tela as coordenadas dos nós de um eixo;*
  - *Axe.checkGeomConsistency(): checa a consistência da geometria de um eixo;*
  - *Node/getDistance(): retorna a distância de um nó para outro;*
- Para um exemplo de um workflow para testes sem a interface gráfica (GUI), veja o arquivo *drive.m*;

To-do's: a modificar ou validar

Formato (importância atribuída segundo minha opinião):
- [Importância][Tipo de tarefa] Título da tarefa;
  - Explicação e outras sub-tarefas;

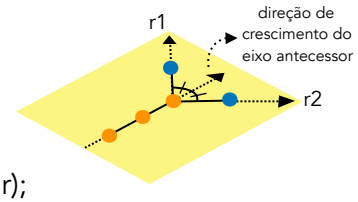
- **[***][Modelo biológico] Validar ou modificar distribuição de axes horizontais e verticais (h/v distribution)**
  - **Explicação:** segundo o artigo de Dupuy, as raízes determinísticas possuem uma distribuição específica de proporção de crescimento horizontal e vertical. Contudo, dado que para raízes com crescimento vertical, suas ramificações podem apenas ter crescimento horizontal, não é possível garantir qualquer distribuição específica (para cada raiz vertical com tamanho suficientemente grande, haverá ao menos uma raiz horizontal). Desta forma, a implementação foi feita de forma que a distribuição final **"tenda"** a distribuição especificada (verificar implementação - *RootStruct/getDetermGrowthDir()*). Nela, esta distribuição é vista como probabilística, no sentido em que ramificações de raízes horizontais têm "*hv_distr(1)*" de probabilidade crescerem horizontalmente e, portanto, "*hv_distr(2)*" de probabilidade de crescerem verticalmente;
  - Há a possibilidade de crescimento determinístico e bifurcação de raízes? Se sim, garantir que estas possuam direção de crescimento vertical ou horizontal apenas?
- **[***][Algoritmo] Problema: é possível criar uma RootStruct com crescimento determinístico sem definir a distribuição h/v;**
  - **Explicação:** para o crescimento determinístico, as únicas direções possíveis são vertical e horizontal. Desta forma, para eixos horizontais, é necessário definir qual a probabilidade de que suas ramificações cresçam horizontal ou verticalmente;
- **[**][Algoritmo/Organizacional] Criar métodos para verificação de validade dos valores dos parâmetros customizados fornecidos pelo usuário;**
  - **Explicação:**
- **[***][Modelo biológico] Validar ou modificar direção de crescimento das ramificações (branch e fork)**
  - **Explicação:** no caso determinístico ou estocástico, as distribuições probabilísticas utilizadas para as direções de crescimento das ramificações são verossímeis? Elas tendem a crescer para baixo?
- **[***][Modelo biológico] Validar ou modificar coeficiente de diâmetro para bifurcação**
  - **Explicação:** o artigo de Dupuy não deixa claro se o coeficiente que relaciona os diâmetros do eixo parente com suas bifurcações é o mesmo coeficiente linear utilizado para o branch. Na implementação atual, é assumido que eles são os mesmos;
- **[**][Modelo biológico] Implementar cálculo do volume em intersecções de eixos;**
  - **Explicação:** Volume calculado no programa será ligeiramente diferente do real (devido as intersecções de diferentes axes - volume da parte visível do Offset node não é calculado);
- **[**][Representação das raízes] Implementar importação/exportação de raízes a partir de um arquivo no formato MTG;**
  - **Explicação:** devido ao fato do formato MTG para representação da base de dados de raízes ser um dos mais populares, a importação dele diretamente no RootGen permite o uso de raízes reais obtidas por medições realizadas por outros pesquisadores. Desta forma, as simulações posteriormente feitas, podem ser mais verossímeis;
  - Ou procurar qual é o formato mais versátil atualmente utilizado - **Sugestão: olhar Root System Markup Language**;

- **[*][Representação das raízes/GUI] Criar GUI para gerar ou importar arquivo no formato escolhido na tarefa anterior a partir do objeto RootStruct;**
- **[**][Geração script] Implementar exportação de um arquivo de dados "brutos" (formato MTG fortemente aconselhável) para padronizar o script python;**
  - **Explicação:** caso haja um arquivo de dados brutos da estrutura radicular, é possível padronizar o arquivo python. Desta forma, não é necessário gerá-lo para cada raiz diferente. O script então iterará sobre esses dados brutos para geração do sólido e simulação no Abaqus. Isto provavelmente melhorará tanto a performance quanto a legibilidade do código python.
- **[*][Geração script] Modificar classe ScriptGenerator para acomodar as modificações feitas na tarefa anterior (ainda é necessário esta classe? Pois script será o mesmo para todas as raízes?);**
- **[*][Organizacional] Definir OffsetNode para axe de ordem 1**
  - **Explicação:** Axe de ordem 1 é o único tipo de axe cujo primeiro nó não corresponde a um OffsetNode;
- **[***][Organizacional/Modelo biológico] Definir OffsetNode para os main laterals**
  - **Explicação:** Como os main laterals
- **[*][Organizacional] Utilizar em todo o programa os atributos *x_glb, y_glb, z_glb* ao invés de utilizar os vetores numéricos (e.g. [0 0 -1]);**
  - **Explicação:** De forma a garantir uma maior versatilidade caso o usuário queira definir outras direções para x, y e z, a implementação começou com as variáveis *x_glb, y_glb* e *z_glb*, mas durante o desenvolvimento, isto foi esquecido e no código algumas vezes as direções foram diretamente colocadas com os valores numéricos;
- **[*][Organizacional] Adicionar os parâmetros *x_glb, y_glb, z_glb* nos atributos da classe Parameters**
  - **Explicação:** após realizar a tarefa anterior, como estes parâmetros podem ser modificados pelo usuário, é necessário que eles estejam na classe Parameters;
- **[*][Organizacional/GUI] Modificar GUIs para que o usuário possa entrar os vetores para *x_glb, y_glb, z_glb***
  - **Explicação:** após realizar a tarefa anterior, é necessário que a partir da GUI o usuário possa entrar com esses valores para serem definidos nos parâmetros;
- **[***][Algoritmo] Possibilidade de não convergência do método numérico** (para determinação do diâmetro inicial): Como as posições do segmentos variam no espaço ao longo das iterações do método numérico, é possível que este não convirja. Tome o seguinte exemplo: definido um dado diâmetro inicial e propagados para todos os nós, se o volume for maior que o especificado, o diametro inicial irá diminuir na próxima iteração. Nela, os axes de maior ordem mudarão para uma posição mais perto do centro do cilindro que define a condição geométrica. Desta forma, novos segmentos de raiz que ultrapassavam estes limites podem não estar mais ultrapassando e, assim, irão contribuir para o volume total da raiz, podendo ser maior do que aquele da iteração anterior (diminuição do diametro inicial => maior volume => funcao não monótona!)

Diretriz Geral da Geração de Raízes

1. **(RootStruct/Constructor) Inicialização (definições e atribuições iniciais da raiz)**
   1.1. Criação da tap root, se aplicável;
   1.2. Definição do global least common ancestor (GlobalLCA);
   1.3. Anexação dos laterals à lista inicial do global LCA;
   1.4. Anexação dos laterals à lista de todos os axes da RootStruct;

2. **(RootStruct/grow()) Crescimento de todos os axes da raiz** - Para testes, utilizar stepGrow() (criação do esqueleto da raiz, sem a atribuição dos diâmetros de cada segmento)
   2.1. Crescimento de todos os axes enquanto ainda houver ativos deles na RootStruct:
      2.1.1. **(Axe/growAxe()) Crescimento do axe atual (se não estiver inativo ("killed"))**
         2.1.1.1. **Incremento** de um segmento de raiz (caso sua distância seja maior do que L_fork, cria um ForkedNode, ao invés de um Node);
         2.1.1.2. **Bifurcação** (fork) do ForkedNode, se aplicável:



            (1) Geração de dois vetores de direção de crescimento r1 e r2 para os dois novos axes (vetor de crescimento dos três axes são coplanares e r1 e r2 são simétricos em relação ao vetor de crescimento do eixo antecessor);
            (2) Teste de condições de contorno para a criação da bifurcação;
            (3) Criação dos novos axes e anexação destes na lista de axes da RootStruct, se aplicável;
            (4) Caso um dos dois novos axes tenha sido efetivamente criado, torna inativo o apex (impede seu crescimento posterior);
         2.1.1.3. **ou Ramificação** (branch), se aplicável (testa se é tap root e pode ramificar e se a distância da última ramificação até o apex é maior que L_branch)
            (1) Cálculo da nova direção de crescimento (a depender se o modelo é estocástico e se o axe corrente é ou não a tap root);
            (2) Teste de condições de contorno para a criação da ramificação;
            (3) Criação do novo axe, anexação deste na lista de axes da RootStruct e atualização da última posição de ramificação, se aplicável;
            (4) Repetição do processo (a partir de 2.1.1.3) para o número de axes de ramificação do modelo (n_branches), caso não seja a tap root;
      2.1.2. **Repetição do processo** (a partir de 2.1.1) para o próximo axe ativo da lista de todos os axes da RootStruct;

3. **(RootStruct/fill()) Preenchimento do esqueleto da raiz**

3.1. Preenche o esqueleto da raiz para que ela possua um volume final especificado, definindo diâmetros para cada segmento. Como o valor do diâmetro é dado por uma função não linear, usa-se um método numérico **(método da bissecção)** para determinar o diâmetro inicial da raiz e, a partir dela, propaga-se os diâmetros de todos os segmentos para calcular o volume da estrutura. Além disso, desloca os axes de ordem maior que 1 de um offset igual ao raio do segmento ascendente, de forma que o crescimento deles se inicie na superfície do cilindro que define o segmento de raiz (ver documento "Notation" e "Filling" para mais detalhes):

3.1.1. Chute inicial do intervalo $\left[d_{min}^{(0)}, d_{max}^{(0)}\right]$ onde a solução (diâmetro inicial) se encontra.

Obs.: o limite superior é calculado supondo que a raiz seja formado apenas pela tap root de diâmetro constante e comprimento *max_depth* com um certo "coeficiente de segurança";

3.1.2. Enquanto não passar de um número máximo de iterações (para evitar a não convergência do método), fazer:

3.1.2.1. Calcular o volume da estrutura radicular para um diâmetro inicial $d^{(k)} := \dfrac{\left(d_{min}^{(k)}, d_{max}^{(k)}\right)}{2}$:

(1) Para cada axe inicial da estrutura radicular:

(a) Atribuição do diâmetro ao primeiro nó do axe;

(b) **(Axe/updateGeometry())** Atualização da geometria do axe (propagação dos diâmetros e atribuição dos offsets)[1];

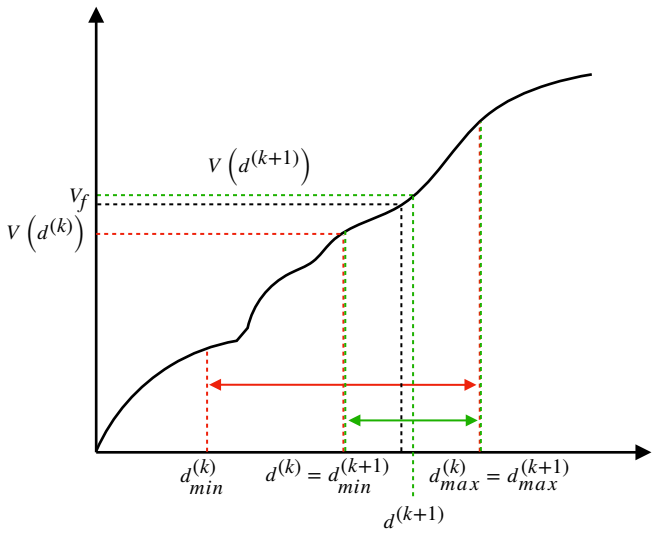
(c) Cálculo do volume do axe e adição no volume total;

3.1.2.2. **(Condição de parada)** Se o erro relativo estiver dentro de uma tolerância, defina o volume final da estrutura como sendo aquele calculado em 3.1.2.1.(1).(c).

Obs.: para evitar erros de truncamento numérico, a condição de parada é implementada da seguinte maneira $\left|V(d) - V_f\right| < tV_f$, onde $V(.)$ é a função não linear que fornece o volume da estrutura em função do diâmetro inicial, V_f é o volume final pré-definido e t é a tolerância;

3.1.2.3. $d_{max}^{(k+1)} \leftarrow d^{(k)}$, se $V\left(d^{(k)}\right) > V_f$ e $d_{max}^{(k+1)} \leftarrow d_{max}^{(k)}$, senão;

3.1.2.4. $d_{min}^{(k+1)} \leftarrow d_{min}^{(k)}$, se $V\left(d^{(k)}\right) > V_f$ e $d_{min}^{(k+1)} \leftarrow d^{(k)}$, senão;



[1] Como o diâmetro varia a cada iteração, é possível que em uma dada iteração, nós não respeitem mais os limites geométricos definidos ou, ainda, que eles possuam um diâmetro menor do que o mínimo especificado.

3.1.2.5. Repita 3.1.2.1;

3.1.3. Caso o método tenha convergido (número de iterações menor do que um máximo especificado):

3.1.3.1. **(RootStruct/prune())** Poda/remove todos os nós inválidos da estrutura (nós cujo diâmetro sejam menores do que o especificado ou que estejam fora dos limites geométricos);