0.1 Nomenclature

- $S_t^\star$ segmentation de I_t, supposée correcte
- $\tilde{S}_{t+1}$ segmentation de I_{t+1} avec la segmentation projetée de I_t (les graines sont les cellules érodées de $S_t^\star$ projetées sur I_{t+1}).
- $Seeds_{t+1}$ image des graines obtenues par sélection à partir de plusieurs images de h-minima
- $\hat{S}_{t+1}$ segmentation de I_{t+1} à partir de $Seeds_{t+1}$

0.2 `_cell_based_h_minima()`

On calcule plusieurs images de h-minima régionaux $Seeds_{t+1}^h$, avec $h \in [h_{min}, h_{max}]$. On calcule d'abord une image de minima qui contient des minima de hauteur comprise entre 1 et h, et on ne sélectionne que les minima de hauteur h avec un seuillage par hystérésis.

On ne conserve que les minima qui sont entièrement inclus dans une seule cellule de $\tilde{S}_{t+1}$. On élimine donc les graines qui chevauchent plusieurs cellules.

Rq: De telles graines renseigneraient pourtant sur le contraste entre 2 cellules adjacentes (sous l'hypothèse que les frontières des cellules dans $\tilde{S}_{t+1}$ soient correctement localisées).

Rq: Dans `ASTEC/ASTEC.py`, le calcul de ces h-minima était fait dans `get_seeds()`, mais la vérification que ces graines étaient effectivement incluses dans les cellules de $\tilde{S}_{t+1}$ était faite dans `get_seeds_from_optimized_paramete` soit après l'appel à `get_back_parameters()`. Potentiellement des graines sélectionnées dans `get_back_parameters()` pouvaient donc être éliminées dans `get_seeds_from_optimized_parameters()`.

0.3 `_select_seed_parameters()`

```
for c, s in n_seeds.iteritems():
    nb_2 = np.sum(np.array(s) == 2)
    nb_3 = np.sum(np.array(s) >= 2)
    score = nb_2*nb_3
    if (s.count(1) or s.count(2)) != 0:
        if score >= tau:
            #
            # obviously s.count(2) != 0
            # the largest h that gives 2 seeds is kept
            #
            h, sigma = parameter_seeds[c][np.where(np.array(s) == 2)[0][0]]
            nb_final = 2
        elif s.count(1) != 0:
            #
            # score < tau and s.count(1) != 0
            # the largest h that gives 1 seeds is kept
            #
            h, sigma = parameter_seeds[c][np.where(np.array(s) == 1)[0][0]]
            nb_final = 1
        else:
            #
            # score < tau and s.count(1) == 0 then obviously s.count(2)) != 0
            # the largest h that gives 1 seeds is kept
```

```
                #
                h, sigma = parameter_seeds[c][np.where(np.array(s) == 2)[0][0]]
                nb_final = 2
            selected_parameter_seeds[c] = [h, sigma, nb_final]
        #
        # s.count(1) == 0 and  s.count(2) == 0
        #
        elif s.count(3) != 0:
            h, sigma = parameter_seeds[c][s.index(3)]
            selected_parameter_seeds[c] = [h, sigma, 3]
        else:
            unseeded_cells.append(c)
            selected_parameter_seeds[c] = [0, 0, 0]
    return selected_parameter_seeds, unseeded_cells
```

Cette fonction est l'équivalent de `get_back_parameters()` dans `ASTEC.py`.

- On a $h_{min} = 4$, $h_{max} = 18$ et on parcourt les h avec un pas δh de 2. L'ensemble des h testés n'est $[h_{min}, h_{max}] \subset \mathbb{N}$ mais $\{4, 6, 8, 10, 12, 14, 16, 18\}$.
- On calcule NB_2 par `nb_2=np.sum(np.array(s)==2)`, c'est le nombre de h qui donnent 2 graines. Elle dépend donc de δh
- On calcule NB_3 par `nb_3=np.sum(np.array(s)>=2)`, c'est le nombre de h qui donnent 2 graines ou plus. Elle dépend aussi de δh. On a évidemment $NB_3 \geq NB_2$

La règle implémentée est

1. S'il existe des h donnant 1 ou 2 graines (la question de la division se pose donc)
   (a) si le score $s(c) = NB_2 \cdot NB_3 \geq \tau$, alors on garde 2 graines. Comme $\tau = 25$, cela signifie que s'il y a au moins 5 valeurs de h qui donnent 2 graines, alors on divisera la cellule.
   (b) sinon ($s(c) = NB_2 \cdot NB_3 < \tau$) et il existe des h donnant 1 graine, alors on garde une graine
   (c) sinon ($s(c) = NB_2 \cdot NB_3 < \tau$ et il n'existe pas de h donnant 1 graine) on garde 2 graines [ce cas doit difficilement survenir, il faut que beaucoup de h ne donnent aucune graine, puis que les suivants donnent au moins 2 graines].
2. sinon (il n'existe pas de h donnant 1 ou 2 graines) et il existe des h donnant 3 graines, alors on garde 3 graines.

   Rq: Toutefois, dans la fonction `get_seeds_from_optimized_parameters()` (dans `ASTEC.py` ou `_build_seeds_from_s` dans `astecnew.py`) créant l'image des graines numérotées à partir des différentes images de h-minima, on ne récupère que les deux premières composantes numérotées ... On crée donc une division, avec un choix artificiel/aléatoire des cellules filles.
   [A CORRIGER ?! par exemple en fusionnant 2 des 3 graines]
3. sinon (il n'existe pas de h donnant 1 ou 2 ou 3 graines), on dit qu'il n'y a pas de graines

   Rq: Pourquoi ne pas considérer le cas 3 graines comme le cas 4 graines ?

On récupère le premier h donnant le nombre choisi de graines. Comme les h sont parcourus par ordre décroissant, c'est donc le plus grand h donnant ce nombre de graines qui est retenu.

0.4 _volume_checking()

```
[...]
    output = _volume_diagnosis(prev_volumes, curr_volumes, correspondences, parameters)
    large_volume_ratio, small_volume_ratio, small_volume_daughter, all_daughter_label = output
[...]
    seed_label_max = max(all_daughter_label)
[...]
    has_change_happened = False
    labels_to_be_fused = []
[...]
    if len(small_volume_ratio) > 0:
[...]
    for mother_c in small_volume_ratio:
[...]
        s = n_seeds[mother_c]
[...]
        nb_2 = np.sum(np.array(s) == 2)
        nb_3 = np.sum(np.array(s) >= 2)
        score = nb_2 * nb_3

        if s.count(1) > 0 or s.count(2) > 0:
            if score >= parameters.seed_selection_tau:
                nb_final = 2
            elif s.count(1) != 0:
                nb_final = 1
            else:
                nb_final = 2
[...]
            if nb_final == 1 and s.count(2) != 0:
                h_min, sigma = parameter_seeds[mother_c][s.index(2)]
                seed_image_name = common.add_suffix(membrane_image, "_seed_h" + str('{:03d}'.format(h_m
[...]
                bb = bounding_boxes[mother_c]
                submask_mother_c = np.zeros_like(prev_seg[bb])
                submask_mother_c[prev_seg[bb] == mother_c] = mother_c
                n_found_seeds, labeled_found_seeds = _extract_seeds(mother_c, submask_mother_c, seed_ima
                if n_found_seeds == 2:
                    new_correspondences = [seed_label_max+1, seed_label_max+2]
[...]
                    selected_seeds_image[selected_seeds_image == correspondences[mother_c][0]] = 0
                    selected_seeds_image[bb][labeled_found_seeds == 1] = seed_label_max + 1
                    selected_seeds_image[bb][labeled_found_seeds == 2] = seed_label_max + 2
                    correspondences[mother_c] = new_correspondences
[...]
            elif (nb_final == 1 or nb_final == 2) and (np.array(s) > 2).any():
[...]
                h_min, sigma = parameter_seeds[mother_c][-1]
                seed_image_name = common.add_suffix(membrane_image, "_seed_h" + str('{:03d}'.format(h_m
[...]
                bb = bounding_boxes[mother_c]
                submask_daughter_c = np.zeros_like(curr_seg[bb])
```

```
                for daughter_c in correspondences[mother_c]:
                    submask_daughter_c[curr_seg[bb] == daughter_c] = mother_c
                submask_mother_c = np.zeros_like(prev_seg[bb])
                submask_mother_c[prev_seg[bb] == mother_c] = mother_c
                aux_seed_image = imread(seed_image_name)
                seeds_c = np.zeros_like(curr_seg[bb])
                seeds_c[(aux_seed_image[bb] != 0) & (submask_daughter_c == mother_c)] = 1
                seeds_c[(aux_seed_image[bb] != 0) & (submask_daughter_c == 0) & (submask_mother_c == mo
[...]
                if 2 in seeds_c:
                    new_correspondences = [seed_label_max + 1, seed_label_max + 2]
[...]
                    for daughter_c in correspondences[mother_c]:
                        selected_seeds_image[selected_seeds_image == daughter_c] = 0
                    selected_seeds_image[bb][seeds_c == 1] = seed_label_max + 1
                    selected_seeds_image[bb][seeds_c == 2] = seed_label_max + 2
                    correspondences[mother_c] = new_correspondences
                    selected_parameter_seeds[mother_c] = [h_min, sigma, 2]
                    seed_label_max += 2
                    has_change_happened = True
[...]
            elif nb_final == 1:
[...]
                selected_seeds_image[selected_seeds_image == correspondences[mother_c][0]] = 0
                selected_seeds_image[deformed_seeds_image == mother_c] = correspondences[mother_c]
                selected_parameter_seeds[mother_c] = [-1, -1, 1]
                has_change_happened = True
[...]
        elif s.count(3) != 0:
[...]
            h_min, sigma = parameter_seeds[mother_c][s.index(3)]
            seed_image_name = common.add_suffix(membrane_image, "_seed_h" + str('{:03d}'.format(h_min))
[...]
            bb = bounding_boxes[mother_c]
            submask_mother_c = np.zeros_like(prev_seg[bb])
            submask_mother_c[prev_seg[bb] == mother_c] = mother_c
            n_found_seeds, labeled_found_seeds = _extract_seeds(mother_c, submask_mother_c, seed_image_
                                                                accept_3_seeds=True)
            if n_found_seeds == 3:
                new_correspondences = [seed_label_max + 1, seed_label_max + 2, seed_label_max + 3]
[...]
                for daughter_c in correspondences[mother_c]:
                    selected_seeds_image[selected_seeds_image == daughter_c] = 0
                selected_seeds_image[bb][labeled_found_seeds == 1] = seed_label_max + 1
                selected_seeds_image[bb][labeled_found_seeds == 2] = seed_label_max + 2
                selected_seeds_image[bb][labeled_found_seeds == 3] = seed_label_max + 3
                correspondences[mother_c] = new_correspondences
                selected_parameter_seeds[mother_c] = [h_min, sigma, n_found_seeds]
                seed_label_max += 3
                has_change_happened = True
                labels_to_be_fused.append(new_correspondences)
```

```
[...]
        else:
            monitoring.to_log_and_console('        .. (8) cell ' + str(mother_c) + ': detected seed numb
                                          + str(s), 2)
[...]

    if len(large_volume_ratio) > 0:
        monitoring.to_log_and_console('      cell(s) with large increase of volume are not processed (y

[...]

    if len(small_volume_daughter) > 0:
        monitoring.to_log_and_console('      process cell(s) with too small daughters', 2)
        for mother_c, daughter_c in small_volume_daughter:
            selected_seeds_image[selected_seeds_image == daughter_c] = 0
            daughters_c = correspondences[mother_c]
            daughters_c.remove(daughter_c)
            if daughters_c:
                correspondences[mother_c] = daughters_c
            else:
                correspondences.pop(mother_c)
        has_change_happened = True
[...]
```

On traite les cellules (mères) ayant eu une forte diminution de volume.
La règle implémentée est

1. S'il existe des h donnant 1 ou 2 graines (les tests suivants sont identiques à ceux de `_select_seed_parameters()`)
   (a) si le score $s(c) = NB_2 \cdot NB_3 \geq \tau$, alors on garde 2 graines. Comme $\tau = 25$, cela signifie que s'il y a au moins 5 valeurs de h qui donnent 2 graines, alors on divisera la cellule ($N_{daughter} = 2$).
   (b) sinon ($s(c) = NB_2 \cdot NB_3 < \tau$) et il existe des h donnant 1 graine, alors on garde une graine ($N_{daughter} = 1$)
   (c) sinon ($s(c) = NB_2 \cdot NB_3 < \tau$ et il n'existe pas de h donnant 1 graine) on garde 2 graines [ce cas doit difficilement survenir, il faut que beaucoup de h ne donnent aucune graine, puis que les suivants donnent au moins 2 graines] ($N_{daughter} = 2$).

   On fait ensuite les opérations suivantes

   (a) Si $N_{daughter} = 1$ et qu'il existe un h donnant 2 graines, on prend les h-minima pour le plus grand h donnant 2 graines, et on recupère ces 2 graines.

   Rq: [Question pour Leo] on crée donc une division là o ù on n'en voulait pas ?!
   (b) sinon si $N_{daughter} = 1$ ou $N_{daughter} = 2$ et qu'il existe un h donnant plus de 2 graines, on prend les h-minima pour le plus petit h (c-à-d h_{min}). Les graines dans $\tilde{S}_{t+1}(mother) \cap \bigcup S(daugther)$, c-à-d les graines à la fois dans la segmentation construite avec les érodés des cellules mères et la segmentation courante sont étiquetées 1, tandis que celles uniquement dans $\tilde{S}_{t+1}(mother)$ (donc pas dans la segmentation courante $\bigcup S(daugther)$ sont étiquetées 2. Potentiellement, ces graines à '2' permettent de récupérer de la matière pour les filles, donc de réduire la diminution de volumes. S'il y a effectivement des '2', avec ces '1' et ces '2', on crée 2 graines pour la segmentation, donc on force $N_{daughter} = 2$.

   Rq: [Question pour Leo] 1) on crée donc une division là où on n'en voulait pas ?! 2) Pourquoi ne pas avoir vérifié, dans le cas précédent, qu'il y avait bien des nouvelles graines donnant potentiellement plus de matière ?!

(c) sinon si $N_{daughter} = 1$ (il n'y a pas de h donnant plus de une graine), on récupère la graine érodée de $\tilde{S}_{t+1}(mother)$

2. sinon (il n'existe pas de h donnant 1 ou 2 ou 3 graines). Cela peut arriver si la cellule fille avait déjà été créée grâce à l'érodé de la cellule mère,

0.5 `astec_process()`

```
[...]
    membrane_image = reconstruction.build_membrane_image(current_time, experiment, parameters,
                                                          previous_time=previous_time)
[...]
```

Reconstruction d'une image de membrane (`reconstruction.build_membrane_image()`), typiquement appel au rehaussement de membrane et fusion avec l'image originale.

```
[...]
    previous_segmentation = experiment.get_segmentation_image(previous_time)
[...]
    undeformed_seeds = common.add_suffix(previous_segmentation, '_undeformed_seeds_from_previous',
                                         new_dirname=experiment.astec_dir.get_tmp_directory(),
                                         new_extension=experiment.default_image_suffix)
    _build_seeds_from_previous_segmentation(previous_segmentation, undeformed_seeds, parameters)
    deformed_seeds = common.add_suffix(previous_segmentation, '_deformed_seeds_from_previous',
                                       new_dirname=experiment.astec_dir.get_tmp_directory(),
                                       new_extension=experiment.default_image_suffix)
    deformation = reconstruction.get_deformation_from_current_to_previous(current_time, experiment,
                                                                          parameters, previous_time)
    cpp_wrapping.apply_transformation(undeformed_seeds, deformed_seeds, deformation,
                                      interpolation_mode='nearest', monitoring=monitoring)
[...]
    if parameters.propagation_strategy is 'seeds_from_previous_segmentation':
        segmentation_from_previous = astec_image
    else:
        segmentation_from_previous = common.add_suffix(membrane_image, '_watershed_from_previous',
                                                       new_dirname=experiment.astec_dir.get_tmp_directo
                                                       new_extension=experiment.default_image_suffix)
    mars.watershed(deformed_seeds, membrane_image, segmentation_from_previous, experiment, parameters)
[...]
```

Calcul de $\tilde{S}_{t+1}$, c'est-à-dire une estimation de la segmentation de I_{t+1} ṗartir de celle de I_t, sans division cellulaire

1. Les cellules de $S_t^\star$ sont érodées pour former l'image de graines S_t^e (`undeformed_seeds`)

   Rq: L'érosion se fait en 26-connexité, avec un nombre d'itérations différent pour les cellules (10) et le fond (25)).

2. Cette image de graines est transformée vers I_{t+1} pour donner $S^e_{t+1\leftarrow t}$ (`deformed_seeds`)

$$S^e_{t+1\leftarrow t} = S^e_t \circ \mathcal{T}_{t\leftarrow t+1}$$

3. On réalise une ligne de partage des eaux pour obtenir l'image $\tilde{S}_{t+1}$ (`segmentation_from_previous`).

```
[...]
    n_seeds, parameter_seeds = _cell_based_h_minima(segmentation_from_previous, cells, bounding_boxes, r
                                                    experiment, parameters)
[...]
    selected_parameter_seeds, unseeded_cells = _select_seed_parameters(n_seeds, parameter_seeds,
[...]
    selected_seeds = common.add_suffix(membrane_image, '_seeds_from_selection',
                                       new_dirname=experiment.astec_dir.get_tmp_directory(),
                                       new_extension=experiment.default_image_suffix)

    output = _build_seeds_from_selected_parameters(selected_parameter_seeds, segmentation_from_previous
                                                   selected_seeds, cells, unseeded_cells, bounding_boxes
                                                   membrane_image, experiment, parameters)

    label_max, correspondences, divided_cells = output
[...]
    segmentation_from_selection = common.add_suffix(membrane_image, '_watershed_from_selection',
                                                    new_dirname=experiment.astec_dir.get_tmp_directory()
                                                    new_extension=experiment.default_image_suffix)
    mars.watershed(selected_seeds, membrane_image, segmentation_from_selection, experiment, parameters)
```

1. `_cell_based_h_minima()` calcule, pour chaque cellule de $\tilde{S}_{t+1}$, le nombre de h-minima (les graines) pour un ensemble de valeurs de h.
2. `_select_seed_parameters()` calcule, pour chaque cellule de $\tilde{S}_{t+1}$, le nombre de graines qui sera retenu (et donc un h optimal). Ce nombre peut varier de 0 à 3.
3. `_build_seeds_from_selected_parameters()` crée l'image des graines $Seeds_{t+1}$ à partir des h sélectionnés auparavant. Ajoute aussi une graine pour le fond.
4. `mars.watershed()` réalise une segmentation de I_{t+1} à partir de $Seeds_{t+1}$. Cette segmentation est notée $\hat{S}_{t+1}$.

- reconstruction d'une image de membrane (`reconstruction.build_membrane_image()`)
- calcul de $\tilde{S}_{t+1}$, c'est-à-dire la segmentation $S_t^\star$ déformée à $t+1$

$$\tilde{S}_{t+1} = S_t^\star \circ \mathcal{T}_{t \leftarrow t+1}$$

- calcul des images de graines Seeds_{t+1}^h pour $h \in [h_{min}, h_{max}]$ (`_cell_based_h_minima()`)
- sélection des paramètres, ie du nombre de graines pour chaque cellule "mère" (`_select_seed_parameters()`)

1 Segmentation de l'image à $t+1$

1.1 h-minima

On calcule plusieurs images de h-minima régionaux $Seeds_{t+1}^h$, avec $h \in [h_{min}, h_{max}]$. On calcule d'abord une image de minima qui contient des minima de hauteur comprise entre 1 et h, et on ne sélectionne que les minima de hauteur h avec un seuillage par hystérésis.

On ne conserve que les minima qui sont entièrement inclus dans une seule cellule de $\tilde{S}_{t+1}$. On élimine donc les graines qui chevauchent plusieurs cellules.

Rq: De telles graines renseigneraient pourtant sur le contraste entre 2 cellules adjacentes (sous l'hypothèse que les frontières des cellules dans $\tilde{S}_{t+1}$ soient correctement localisées).

1.2 `get_seeds_from_optimized_parameters()`

La fonction `get_seeds_from_optimized_parameters()` (dans `ASTEC.py` ou `_build_seeds_from_selected_parameters()` dans `astecnew.py`) construit une image de graines à partir des paramètres retenus dans `get_back_parameters()`.

- 1 graine: on récupère la graine dans $Seeds_{t+1}^h$ pour la cellule c
- 2 graines: on récupère les 2 graines dans $Seeds_{t+1}^h$ pour la cellule c
- 3 graines: on ne récupère que 2 graines (les deux premières numérotées) dans $Seeds_{t+1}^h$ pour la cellule c.

  Rq: Ce comportement doit être corrigé.

- 0 graine: on regarde si le volume de la cellule c (dans $\tilde{S}_{t+1}$) est suffisamment grand (supérieur à 100). Si oui, on récupère alors la graine correspondante dans $S_{t+1 \leftarrow t}^e = S_t^e \circ \mathcal{T}_{t \leftarrow t+1}$ (cellules de $S_t^\star$ érodées (10 itérations en 26-connexité) puis transformées).

  Rq: Il y a ici une potentielle fin de linéage.

- fond: on récupère toutes les graines de $Seeds_{t+1}^{h_{min}}$ qui correspondent à la cellule 1 (le fond).

On

1.3 `volume_checking()`

Cherche à corriger des erreurs détectées par des changements de volume trop importants.

Pour une cellule c^t_i

[1]

References

[1] Léo Guignard. *Quantitative analysis of animal morphogenesis: from high-throughput laser imaging to 4D virtual embryo in ascidians.* Theses, Université Montpellier, December 2015.