Pseudo-code for CHANS Modeling

Model Initiation

The *Create-environment* function reads in external data to form the model environment (world).

```
Create-environment()
[
    Read external data and assign environmental and geographic data pixels;
]
```

The *Create-agent* function reads in external data to form the model agents. The set attributes functions referred to here are defined below.

```
Create agents()
[
    Set-person-attributes for all person agents;
    Set-household-attributes for all household agents;
]
```

Setup Agent Attributes

Set-person-attributes is a 3rd–level procedure, called by the initiation process and several 2nd level procedures. *min-birth-age*, and *max-birth-age* could be set as global variables that have constant values for all agents and all times.

```
Set-person-attributes (agent, seed-agent = null, type)
[
If (type = “initiation”) then
    ** Set up attributes based on real individual data through reading
    external files.
    1) empirical histogram #1 is percent of married women having 0, 1, 2,
       …, over 6 children;
    2) empirical histogram #2 is histogram of time steps between marriage
       and bearing 1st child;
    3) empirical histogram #3 is histogram of time interval between
       consecutive births **

    Set personal ID, age, education, address, local residence, etc. from
    reading external file;

    ** Below set up attributes based on aggregate data or regression**
    Set birth-plan as function (age, education, etc.) or empirical
    histogram #1;

    Set first-birth-interval as function (age, education, etc.) or
    empirical histogram #2;

    Set birth-interval as empirical histogram #3;

else if (type = “new birth” )    then    ** for new babies **
    Set education as the max of father and mother (seed-agent)’s education
    or function (sex, local environment, etc.) ;
    Set unique personal ID and household ID as mother (seed-agent)’s
    household ID;

    Set address or local residence as mother (seed-agent)’s address;

else if (type= “marriage”)   then
    Set birth-plan, birth-interval using histogram or use regression
    models;

    Set first birth time = marriage time + first-birth-function;

    Set address or local residence as husband (seed-agent)’s address;

else if (type = “in-migrants” ) then
    Set education as the seed-agent’s or from empirical histogram or use
    education model;

    Set unique personal ID;

If (agent is female AND agent is married) then
    Set birth-plan,  birth-interval to equal the seed-agent’s or from
    empirical histogram

    Set address or local residence as seed-agent’s address;

    If (agent has no children) then
        Set first birth time = marriage time + first-birth-function;
]
```

Set-household-attributes is used to initialize new households in the model, whether at model initialization (time zero) or when new households form due to in-migration.

```
Set-household-attributes (household agent, seed-household-agent = null,
type)
[
If (type = “initiation”) then
    ** Below set up attributes based on real household data through reading
    external files**

    Set household ID and address or local residence, etc. from reading
    external file;

If (type = “in-migration”) then
    Set address or local residence, etc.  from seed-household-agent;

    Set unique household ID;  **ID not reused**

    Set household size X from seed-household-agent;

    Create X person agents;

    Set-person-attributes for these X agent;

If (type = “new household establishment”) then
```

```
    Set address or local residence, etc. from seed-household-agent
    (parent’s household);

    Set unique household ID;  **ID not reused**
]
```

Major process

Model-loop is the main loop in the model. It determines the order in which events occur in the model. The loop runs through time steps 1, 2,…, N (N is the simulation time span). Sometimes this loop is implicit—e.g., time-ticks automatically move and we do not see an explicit loop.

```
Model-loop
[
    Loop through all households
    [
        Check household sizes; ** eliminate households with size=0 **
        Migrate-out-household-level;
        Migrate-in-household-level;

        Loop through all potential new households within the household:
        [
            For each potential household agent:
                If (will-establish-household is true)
                [
                    Establish-household;
                ]
        ]

        Loop through all marriages within the household:
        [
            ** Potentially subject to divorce with a small probability**
            Divorce;
        ]

        Loop through all persons within the household:
        [
            Grow (die, or increment age);

            If (person agent is qualified-for-marriage) then
                Marry;

            If (person agent is qualified-for-birth)  then
                Give-birth;

            If (person agent is qualified-for-outmigration)
                Migrate-out-individual-level;

            If (person agent is qualified-for-in-migration)
                Migrate-in-individual-level;
        ]
    ]
]
```

Logical tests

will-establish-household is used to test whether a new household will form out of a "potential household".

```
will-establish-household
[
    Household-establish probability = function (household agent, local
    environment, potential new household attributes) or empirical
    histogram;

    If (size of household-agent > minimum-household-size AND random # <
    Household-establish probability) then
    [
        Set household to be will-establish-household;
    ]
]
```

qualified-for-marriage is used to test whether a person is able to get married at a given timestep (they must be old enough, single, etc.).

```
qualified-for-marriage
[
    if (Person agent is single AND agent age > min-marry-age AND agent age
    < max-marry-age) then
    [
        Set person agent to be marriage-qualified
    ]
]
```

qualified-for-birth is used to test whether a person is able to give birth at a given timestep (they must be old enough, not have already had beyond their desired number of children, etc.).

```
qualified-for-birth
[
    If (person agent is female and married AND person’s age > min-birth-age
    AND person’s age < max-birth-age AND number of children < birth-plan)
    then
    [
    Set person agent to be birth-qualified;
    ]
]
```

qualified-for-outmigration is used to test whether a person will out-migrate at a given timestep.

```
qualified-for-outmigration
[
    If (person agent’s age > min-outmigration-age AND person age <
    max-outmigration-age) then
    [
        Set person agent to be outmigration-qualified
    ]
]
```

qualified-for-in-migration is used to test whether a person will in-migrate at a given timestep.

```
qualified-for-in-migration (seed-person)
[
    if (seed person age > min-in-migration-age AND person age <
    max-in-migration-age) then
    [
        Set person agent to be in-migration-qualified
    ]
]
```

Agent Actions

Grow increments an agents age by one timestep.

```
Grow (agent)
[
    Mortality-probabilities = function (agent, local environment) OR
    mortality histogram by age/sex;

    If (random #s match mortality-probabilities) then
    [
        Agent dies;
        Remove agent from agent-list;
    ]

    Else
    [
        Agent remains;
        Agent’s age grows by 1; ** Age = Age + 1 **
    ]
]
```

Migrate-out-individual-level makes an individual migrate. Local environment could be agent’s residence; min-migration-age and max-migration-age could be set as global variables that have constant values for all agents and at all times.

```
Migrate-out-individual-level (agent, agent-list, local environment)
[
    outmig-probabilities = function (agent, local environment) OR
    out-migration histogram by age/sex;

    If (random #s agree match outmig-probabilities) then
    [
        Agent migrates out; ** Remove agent from agent-list **
    ]
    Else
    [
        Agent remains; ** Agent remains on agent-list **
    ]
]
```

Migrate-out-household-level makes an entire household migrate.

```
Migrate-out-household-level (agent, household, household-list)
[
    Set whole-household-out-mig-probability to be function (agent, local
    environment) OR whole household migration histogram;

    if (random # < whole-household-out-mig-probability)
    [
        Remove-household (agent, household, household-list);
    ]
]
```

Migrate-in-household-level makes an entire household migrate in.

```
Migrate-in-household-level (seed agent, household-list, local environment)
[
    inmig-probability = function (local environment, seed-agent attributes)
    OR in-migration histogram by age/sex;

    If (random # < inmig-probability) then
    [
        Create a new household;
    ]

    Set-household-attributes for this agent using seed-agent;
]
```

Marry makes two agents marry each other. *min-marry-age* is set as a global variable with a constant value for all agents and at all times. Below we only *S1* to be a female.

```
Marry (agent S1, single-male-list, single-female-list, local environment)
[
    female-marry-probabilities = function (agent S1, local environment) OR
    female marry histogram by age;

    If (agent S1 is female AND random #s match female-marry-probabilities)
    then
    [
        If (S1 is single and random # < empirical probability) then
        [
            ** Allow marrying an in-migrant **
            Set S1 status to married;
            Create new in-migrant S2;
            Remove S1 from single-female-list;
        ]

        Else loop through single-male-list:
        [
            Find a random agent S2 from single-male-list;
            Calculate age difference between S1 and S2;
            Calculate ethnicity difference between S1 and S2;
            Calculate xxx difference between S1 and S2;
            Marry-probability = product of probabilities due to age,
            ethnicity, and other Differences

            If (random # < Marry-probability) then
            [
                Set S1 status to married;
                Remove S1 from single-female-list and S2 from
                single-male-list;
                Get out of loop;
            ]
        ]
        If (S1 is married) then
        [
            Change S2’s status to married;
            Set S1’s spouse to be S2 and S2’s spouse to S1;
            Set S1 and S2’s marriage age to be their current age;
        ]
    ]
**So far this method does not guarantee marriage at this step**
]
```

Divorce makes two agents marry each other. Divorces can occur with a probability set as the probability of a divorce occurring for a marriage in a given timestep OR they can be modeled with a probability that is calculated for each spouse (for example a wife may be more likely to divorce her husband than for her to divorce him, or vice versa). Here we use the simple model of probability of a marriage ending in divorce in a given timestep. Therefore we only want to consider each marriage ONCE, so loop through the agents, and make sure that we do not consider divorce for both spouses, only the first we see in the loop (as the probability of divorce in this simple case is independent of the spouses characteristics. **

```
Divorce (agent, household, household-list)
[
    divorce-probability = function (local environment, agent attributes) OR
    divorce-histogram by age/etc.;
    If (random # < divorce-probability) then
    [
        Add husband to single-male-list and wife to single-female-list;
        Change marriage status of husband and wife to single;
        Establish-household with wife as move-out-person and type equal to
        “divorce”;
    ]
]
```

Give-birth makes an agent given birth. *min-birth-age*, *max-birth-age* could be set as global variables that have constant values for all agents and all times.

```
Give-birth (agent, min-birth-age, max-birth-age)
[
    if (agent has zero child) then
    [
        female-1st-kid-probability = function (agent, local environment) OR
        female-1st-kid-histogram by age;
    ]
    If (random # < female-1st-kid-probability) then
    [
        Set first-birth to TRUE;
    ]
    if ((first-birth is TRUE) OR (agent has one or more children AND
    current time – time of last birth >= birth interval)) then
    [
        ** for bearing 2nd or later child **
        create an agent (child);
        Set-person-attributes (child, type= “birth”);
    ]
]
```

Migrate-in-individual-level makes an agent in-migrate. Local environment could be agent’s residence; *min-seed-agent-age* and *max-seed-agent-age* could be set as global variables that have constant values for all agents and at all times.

```
Migrate-in-individual-level (seed-agent, agent-list, local environment)
[
    inmig-probability = function (local environment, seed-agent attributes)
    OR in-migration histogram by age/sex;
    If (random # < inmig-probability) then
    [
        Create a new agent;
        Set agent age to be close to seed-agent;
        Set-person-attributes for this agent using seed-agent;
    ]
]
```

Establish-household forms a new household.

```
Establish-household (seed-household, move-out-persons, type,
household-list, local environment)
[
    Create a new household;
    If (type = “household establishment”) then
    [
        ** This clause is for cases like a new household being established
        by a married couple **
        Set the location of the new household within or near
        seed-household;
    ]
    Else if (type = “divorce’) then
    [
        Set the location of the new household to be randomly located;
        Set other attributes of the new household to equal (be close to)
        those of the Seed-household;
        Add this new household to the household-list;
        Remove move-out-persons from seed-household-agent;
        Add move-out-persons to new household;
    ]
]
```

Check-household-sizes checks households to ensure they are not empty. Any empty households are removed from the model.

```
Check-household-sizes (household-list)
[
    Loop through all household agents:
    [
        If (the size of household = 0) then
        [
            Remove-household (household, household-list)
        ]
    ]
]
```

Remove-household removes a household agent from the model.

```
Remove-household (household, household-list)
[
    Remove household from the household-list;
]
```

Set-first-birth-function sets the first birth function for an agent. This function relies on *min-1st-birth-model*, a global variable, which may take values of “simple”, or “histogram”, or “regression” **

```
Set-first-birth-function ()
[
    If (min-1st-birth-model is equal to “simple”) then
    [
        Set first-birth-function to be 12 months (or 1 year);
    ]
    Else if (min-1st-birth-model is equal to “histogram”) then
    [
        Set first-birth-function to be a number drawn from histogram;
    ]
    Else if (min-1st-birth-model is equal to “regression”) then
    [
        Set first-birth-function to be equal to function (local
        environment, marriage age, etc.);
    ]
]
```