Informe técnico final

Anexo 11

Documentación del sistema para la estimación de saldos netos migratorios

1. Documentación del sistema para la estimación de saldos netos migratorios

1.1 Introducción

El sistema para la estimación de saldos netos migratorios o el sistema ESNM tiene como objetivo implementar el algoritmo propuesto en este proyecto para estimar los saldos netos migratorios (ASNM) para cada una de las entidades federativas de México en los periodos intercensales.

Este documento tiene como objetivo presentar y describir las principales partes del código fuente del sistema ESNM. El documento sirve como un apoyo adicional para entender el código. Se recomienda revisar el código fuente y este documento simultáneamente.

Es importante señalar que el código fuente fue realizado en el idioma inglés, por lo que las referencias a nombres de variables, clases, métodos, funciones y otros componentes están definidos en dicho idioma. Estos están resaltados con letra *itálica* a lo largo del documento. Por ejemplo, la clase *StatesPopulation* está en itálica y en inglés, representado la clase que proporciona el vector de la población por entidades federativas. Como apoyo a la documentación del software, se hace uso de diagramas de clases del lenguaje unificado de modelado (UML, por sus siglas en inglés).

El proyecto combina el uso de programación orientada a objetos y programación procedural debido a que es una practica usual en el lenguaje seleccionado (ver 1.2.1 Lenguaje Python). Es debido a esto que en los diagramas de clases se utiliza la etiqueta "*utility*" para organizar un grupo de funciones que estrictamente hablando no son una clase. En realidad se trata de archivos o módulos de Python que definen varias funciones.

A continuación se describen las herramientas de desarrollo de software sobre las cuales se ha trabajado (sección 1.2 Herramientas de desarrollo de software). Después se presentan cada uno de los componentes o aplicaciones que dan forman al sistema ESNM (sección 1.3 Diseño de Sistema ESNM). Y por ultimo, se describen los métodos para condensar e ingresar datos adicionales al sistema (sección 1.4 Condensación y Preparación de datos).

1.2 Herramientas de desarrollo de software

A continuación se describen y justifican el lenguaje de programación seleccionado, paquetes de apoyo y otras herramientas.

1.2.1 Lenguaje Python

Para este proyecto se utilizó el lenguaje de programación Python, un lenguaje de alto nivel y propósito general que es usado en aplicaciones científicas. Algunas de las bondades de Python son su facilidad de aprender su sintaxis, la rapidez con la que se pueden desarrollar aplicaciones, el gran número de bibliotecas o módulos con los que cuenta desde su instalación, entre otras cosas (Hettinger, 2012).

Python es el lenguaje seleccionado para la programación de tareas en aplicaciones para el análisis estadístico y geo estadístico como lo son SPSS y ArcGIS respectivamente.

Python permite el desarrollo rápido de aplicaciones web con el uso de *frameworks* como Django o Flask. Un *framework* para el desarrollo de aplicaciones web proporciona componentes frecuentemente necesarios en un solo paquete y se encarga de la integración de los mismos con poca participación del desarrollador (Bennett, 2008). Ejemplos de estos componentes o tareas, son los que se encargan de mostrar platillas HTML o de realizar consultas a las bases de datos. Esto permite que las aplicaciones programadas para el análisis de datos puedan tener una presentación de tipo web rápidamente y sin necesidad de usar un lenguaje de programación adicional.

1.2.2 Herramientas de Matemáticas y Estadística

Algunas de las bibliotecas más importantes (llamadas módulos en Python) que se utilizan en este sistema son NumPy, SciPy y Pandas. NumPy permite el manejo de grandes arreglos numéricos y matrices de forma eficiente y conveniente.

SciPy consiste de un conjunto de funciones de alto nivel implementadas sobre NumPy. Por ejemplo, SciPy implementa diferentes métodos de interpolación como el método de splines cúbicos.

Pandas es otro módulo implementado sobre NumPy que define estructuras de datos que facilitan el manejo de vectores, conjuntos de vectores y matrices. Principalmente define las estructuras *Series* y *DataFrame*. *Series* es una mejora a los arreglos numéricos de Numpy que permite incluir un índice especifico que no necesariamente tiene que iniciar en 0 o ser números enteros. Por ejemplo, podemos definir un índice de 1 a 32 para representar a cada una de las entidades de México, o incluso crear un índice de con letras o cadenas alfanuméricas. *DataFrame* puede interpretarse como un conjunto de elementos

del tipo *Series*. Esto permite trabajar con grandes grupos de datos usando la misma lógica que en hojas de cálculo o incluso aplicar funciones matriciales.

1.2.3 Django

El sistema ESNM se desarrolló con la idea de permitir al usuario generar reportes y gráficas de forma interactiva con tecnologías web (HTML, Javascript, CSS). Para agilizar el desarrollo del sistema se utilizó Django, un *framework* de aplicaciones web que usa el lenguaje Python. Django simplifica el desarrollo de aplicaciones web proporcionando bibliotecas de programación que realizan tareas que frecuentemente son necesarias en este tipo de proyectos.

1.2.4 Gráficas y visualizaciones: D3.js

D3.js es la principal biblioteca de JavaScript utilizada como apoyo para la elaboración de gráficas y visualizaciones (véase Figura 1). D3.js permite la visualización de gráficas y cartografías en el navegador sin necesidad de crear imágenes y permitiendo crear representaciones interactivas. Por ejemplo, se pueden cambiar la fuente de los datos en una gráfica de forma dinámica e interactiva, permitiendo reflejar cambios en la pantalla sin necesidad de descargar una imagen diferente o recargar la página completa.

Para agilizar más la elaboración de gráficas se utilizó la biblioteca NVD3 que permite reutilizar el código de graficas desarrolladas sobre D3.js (véase Figura 1).

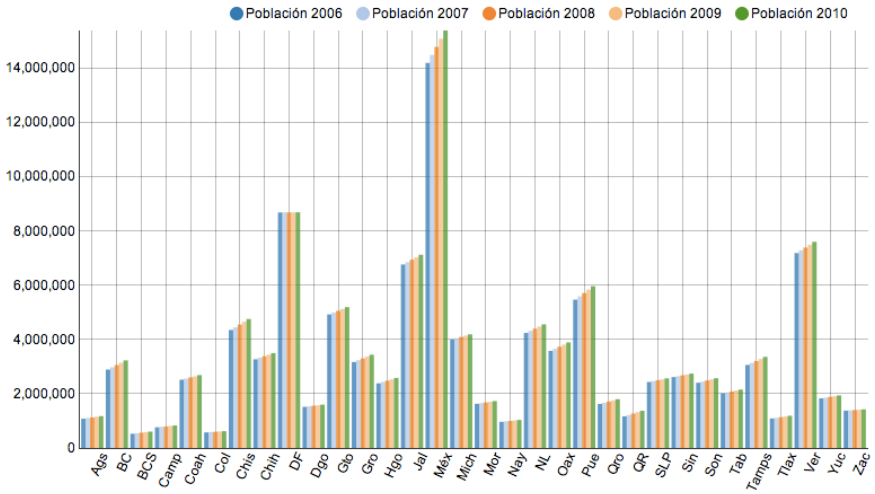

Figura 1. Ejemplo de gráfica usando NVD3

1.2.5 Otras herramientas utilizadas

Otra herramienta utilizada a lo largo del sistema es el *framework* de presentación al usuario llamada Bootstrap. Esta herramienta define la apariencia, tipo y colores de los textos, imágenes, botones y otros componentes gráficos. Esto simplifica el diseño y desarrollo de las interfaces de usuario de las aplicaciones web.

1.3 Diseño de Sistema ESNM

El sistema ESNM fue desarrollado sobre Django y en esta sección se describen su partes fundamentales. Una forma de introducir y conceptualizar el diseño del proyecto es ver su estructura de directorios y archivos. Esta estructura puede visualizarse en la Figura 2.

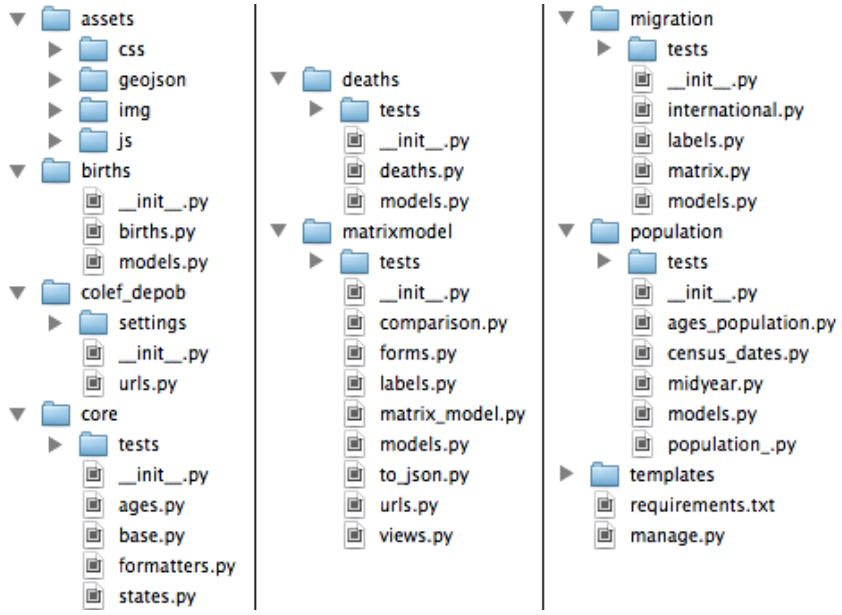

Figura 2. Estructura del proyecto

En Django, un proyecto se forma de archivos de configuración, aplicaciones, bibliotecas, archivos estáticos y plantillas. El nombre del proyecto es *colef_depob*, y bajo un subdirectorio del mismo nombre se encuentran los archivos de configuración. Dentro de este subdirectorio, el archivo *settings.py* es el que contiene ajustes importantes como la configuración de la base de datos, el registro de aplicaciones que se usan en el proyecto, además, se definen la ubicación de directorios de archivos estáticos, la zona horaria y el idioma.

El archivo *manage.py*, ubicado en el directorio raíz del proyecto, es un guión o *script* que se encarga de tareas comunes para la administración del proyecto. Se utiliza para iniciar el servidor, preparar la base de datos, entre otras cosas.

El directorio *assets* contiene los archivos estáticos que no son generados o modificados por los usuarios. Esto incluye los archivos de estilo (*css*), archivos para visualización geográfica (*geojson*), imágenes (*img*), scripts y bibliotecas del lenguaje JavaScript (*js*).

El directorio *templates* contiene las plantillas de tipo *HTML* que definen la estructura de presentación para cada pagina o sección de la aplicación web. Estos archivos son una combinación del lenguaje *HTML* y el lenguaje de plantillas proporcionado por Django, que es una versión simplificada de Python.

En Python, un módulo es un archivo que contiene definiciones y sentencias de código. En Django, el proyecto puede ser organizado en directorios que agrupan archivos o módulos que tienen un propósito en común. Estos directorios de módulos se llaman aplicaciones. Una aplicación es una biblioteca pequeña que ha sido diseñada para representar un aspecto único de un proyecto (Greenfeld y Roy, 2013). Las características con las que se pueden identificar las aplicaciones en el proyecto, son los directorios que contienen los archivos *__init__.py*, *models.py* o *urls.py*, con excepción del subdirectorio *colef_depob*. Este proyecto está formado de 6 principales aplicaciones con funciones especificas cada uno: *core*, *population*, *births*, *deaths*, *migration* y *matrixmodel*. A continuación se describen cada una de estas aplicaciones.

1.3.1 Aplicación de módulos de utilidad (*core*)

En esta aplicación o paquete se definen algunos módulos, clases (véase Figura 3) y funciones de propósito general que son útiles en varias partes del sistema. Esto permite mantener de forma organizada las piezas de código que no pertenecen a una aplicación particular y facilita su uso cuando son necesitadas.

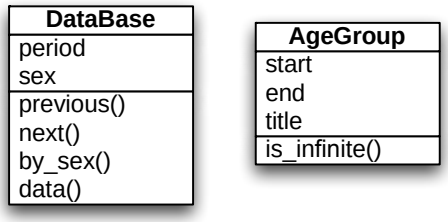

Figura 3. Diagrama de clases: *core*

1.3.1.1 base.py

El módulo *base* define la clase abstracta *DataBase* (véase Figura 3) que proporciona la estructura general necesaria para identificar y almacenar un vector de datos. Define los atributos para almacenar el periodo (*period*) y el sexo (*sex*) de los datos.

También se especifican los métodos abstractos *previous*, *next* y *by_sex* para obtener objetos relacionados de forma conveniente. El método *next* debe inicializar un objeto del mismo tipo con los mismos parámetros pero para el periodo posterior (*e.g.* 5 años después, si hablamos de población). El método *previous* es muy parecido pero debe inicializar el objeto para el periodo anterior (*e.g.* 5 años antes). El método *by_sex* debe inicializar dos objetos del mismo tipo y con los mismos parámetros pero para ambos sexos, devolviendo una tupla de vectores, uno de hombres y otro de mujeres.

En este módulo también se definen las constantes *MAX_STATE*, *MEN* y *WOMEN* que son utilizadas en diferentes partes de este sistema. *MAX_STATE* (32) se puede interpretar como la cantidad de entidades federativas del país o el número identificador del último estado. *MEN* (1) y *WOMEN*(2) identifican los datos de hombres y mujeres respectivamente. Los valores 1 y 2 fueron seleccionados en base a bases de datos del Instituto Nacional de Estadística y Geografía (INEGI).

1.3.1.2 ages.py

El módulo *ages* incluye objetos, funciones y constantes que son necesarios para el manejo de grupos de edad en el proyecto. Estos permiten que vectores de población, nacimientos, defunciones, migración, entre otros, puedan ser agrupados por rangos de edad.

La clase *AgeGroup* (véase Figura 3) permite especificar el rango de edad o grupo de edad. Sus parámetros de inicialización son *start* que indica la edad inicial y *end* la final o tope. El parámetro *end* acepta el string “inf” para indicar que el grupo no tiene un valor tope, es decir, representa el grupo con una edad inicial dada hasta el infinito.

Se pueden formar listas de objetos de tipo *AgeGroup* que sirven como índices y permiten manejar la información en vectores de datos agrupados por rangos de edad. La función *make_equidistant_groups* permite generar una secuencia de grupos de edad equidistantes, es decir, grupos de edad donde la distancia entre la edad inicial de cada grupo se mantiene para la edad inicial del grupo subsecuente. Por ejemplo, se puede generar una lista de grupos de edad con una distancia 5 para obtener la población del país por grupos quinquenales (*e.g.* 0-4, 5-9, …, 95-99, 100+).

Una de las constantes definidas en este módulo son *INF_VALUE* (199) que sirve como un valor infinito que permite representar un grupo de edad sin tope. *MISSING_AGE* (999) es otra constante para identificar a los registros sin edad registrada.

1.3.1.3 states.py

El módulo *states* define principalmente los arreglos de cadenas de texto que contienen los nombres de las 32 entidades federativas (*STATES*), sus abreviaciones (*ABBR*) y sus claves de 2 caracteres (*KEY*).

1.3.2 Aplicación de Población (*population*)

Esta aplicación o paquete se encarga de preparar los datos de población de Censos y Conteos del país. Los datos de población se pueden obtener por entidad federativa, sexo o grupos de edad con ajustes de prorrateo, suavizado y mitad de año, entre otros. A continuación se describen cada uno de los módulos que conforman esta aplicación (véase Figura 4).

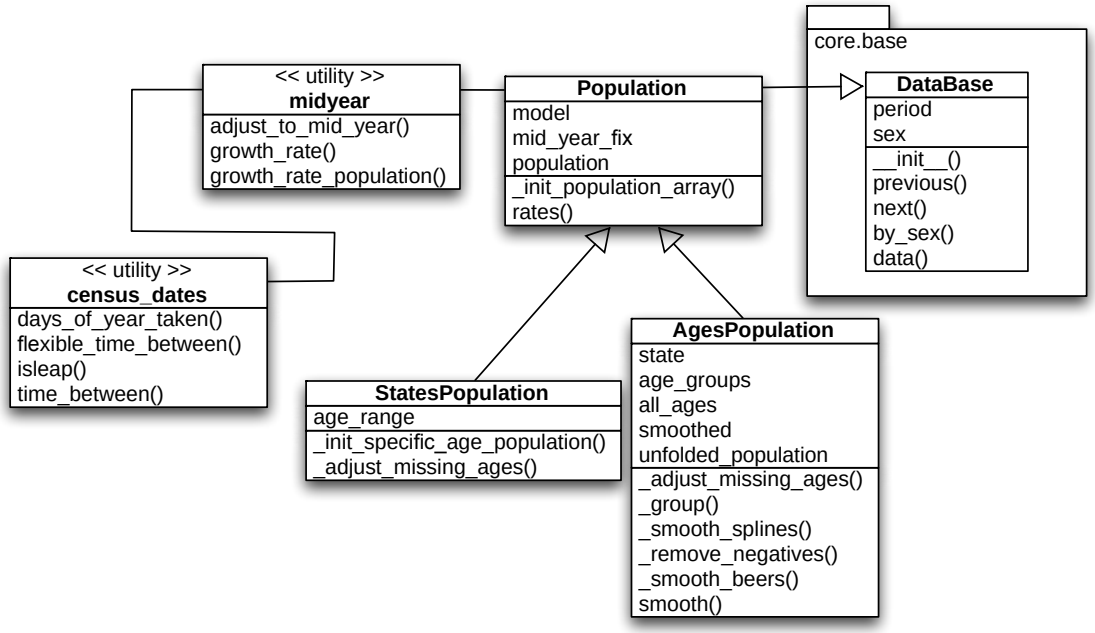

Figura 4. Diagrama de clases: *population*

1.3.2.1 population_.py

Este módulo incluye la clase abstracta *Population* y la subclase *StatesPopulation* (véase Figura 4). La primera proporciona una estructura general con los atributos que ayudan a identificar y almacenar un vector de población. La

segunda, *StatesPopulation*, es una subclase de *Population* que proporciona y prepara el vector de poblaciones por entidad federativa para un periodo dado.

Los dos principales atributos que se proporcionan en el método de inicialización de la clase abstracta Population son el identificador de sexo (*sex*) y el periodo (*period*). Este identificador de sexo se define como 1 para hombres y 2 para mujeres. Se hace uso de las constantes *MEN* y *WOMEN* que contienen ya estos valores para una mejor documentación del código. En dado de no proporcionar el identificador de sexo o que éste sea cero (0), la población obtenida representara a ambos sexos. En el caso del periodo, solo se permite introducir periodos de los censos y conteos que se encuentran almacenados en el sistema (*e.g.* periodo 2010 para obtener las poblaciones censadas para cada entidad federativa en el 2010). El método *rates* permite obtener el vector de tasas de población con respecto a la población total.

Por otro lado, el método de inicialización de la clase *StatesPopulation* recibe también un rango de edades (*age_range*). El rango de edades (*age_range*) debe ser un arreglo de edades especificas con las que se desea trabajar. Por ejemplo, si se desea obtener la población por entidades federativas de solo los menores de edad del país (menores de 18 años de edad), se debe proporcionar un arreglo que contenga los valores de 0 a 17. En dado caso que se utilice el rango de edades, la clase *StatesPopulation* se encarga de realizar un ajuste de prorrateo para incorporar los casos sin edad registrada. Si el rango de edad no es proporcionado o es nulo (*None*), todas las edades se toman en cuenta y el tiempo de respuesta es menor.

Después de crear una instancia de *StatesPopulation*, se puede acceder al vector de poblaciones desde el atributo *population*. Este vector se forma de 32 poblaciones, una por cada entidad federativa del país y es una estructura de tipo *Series* del módulo *Pandas* (véase 1.2.2 Herramientas de Matemáticas y Estadística), que permite aplicar diferentes operaciones matemáticas y mantener un índice especifico.

1.3.2.2 age_population.py

Este módulo incluye la clase *AgesPopulation* (véase Figura 4), otra subclase de *Population* que proporciona y prepara un vector de poblaciones por edad o grupos de edad para una entidad federativa, sexo y periodo específicos.

El método de inicialización de *AgePopulation* puede recibir el periodo, grupos de edad, estado y sexo. El parámetro de grupos de edad (*age_groups*) debe ser un arreglo de objetos de tipo *AgeGroup* (véase 1.3.1.2 *ages.py*). Cada uno de estos objetos contiene el inicio y final de un rango de edades. Por ejemplo, si se desean grupos quinquenales, terminando en la edad 100, se pueden definir los grupos de edad 0-4, 5-9, 10-14, …, 90-94, 95-99 y 100+. El ultimo grupo representa el rango de edades con inicio en 100 e infinito como edad máxima

(sin edad máxima), la cual se usa para acumular la población de 100 años de edad en adelante. En caso de no proporcionar el arreglo de grupos de edad o que sea nulo (*None*), el vector de poblaciones será de tamaño 141 con las poblaciones para todas las edades de cero (0) a 140.

Cuando hablamos de poblaciones por edad debemos de tomar en cuenta el ajuste de prorrateo que reparte proporcionalmente la población que no registro su edad. El método privado *_adjust_missing_ages* en la clase *AgePopulation* se encarga de aplicar internamente este ajuste al vector de población.

El método privado *_group* de la clase AgesPopulation, se encarga de agrupar los valores del vector de población en los grupos de edad indicados en el arreglo *age_groups*. Cabe mencionar que la estructura de datos *Series* (de la biblioteca Pandas) permite que se tome como índice del vector resultante de población el mismo arreglo *age_groups*.

El método público *smooth* permite suavizar los valores del vector de población por edad. Internamente, smooth puede manejar 2 métodos distintos de suavizado. Estos son la interpolación por splines cúbicos (*_smooth_splines*) y el método de coeficientes de Beers (*_smooth_beers*).

Al igual que con la clase *StatesPopulation*, se puede acceder al vector de poblaciones de *AgesPopulation* con el atributo *population*. Este vector se forma de N poblaciones, donde N representa el numero de grupos de edad con el que fue inicializado el objeto.

1.3.2.3 midyear.py

En este módulo se define la función *adjust_to_mid_year* que ajusta el vector de población a mitad de año utilizando la tasa de crecimiento aritmético en base al periodo anterior. Por ejemplo, para calcular la población ajustada del 2010, se calcula la tasa de crecimiento entre 2005 y 2010, con los datos del conteo 2005 y el censo 2010. Con esta tasa de crecimiento estimamos la población a mitad de año del 2010, es decir el 1 de Julio de 2010. La función que calcula la tasa de crecimiento aritmético se llama *growth_rate.*

1.3.3 Aplicación de Nacimientos (*births*)

La aplicación *births* se encarga de preparar y proporcionar los datos de nacimientos y tasas de natalidad. Estos datos fueron obtenidos principalmente de las bases de datos de nacimientos del Sistema Nacional de Información en Salud de 1985 a 2011. La datos de nacimientos se pueden obtener por entidad federativa (entidad de residencia habitual) o por edad de la madre (en el momento del nacimiento).

También se cuenta con las tasas de natalidad publicadas por el Consejo Nacional de Población de 1990 a 2010.

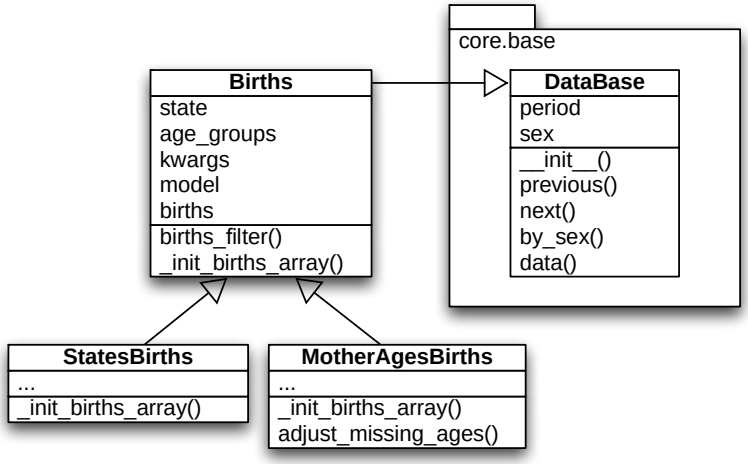

Figura 5. Diagrama de clases: *births*

1.3.3.1 births.py

El módulo *births* (de la aplicación del mismo nombre) define la clase abstracta *Births* que proporciona una estructura general con atributos que ayudan a identificar y almacenar un vector de nacimientos. *Births* es una subclase de *DataBase* (véase 1.3.1.1 *base.py*). De la clase *Births* se derivan las subclases *StatesBirths* y *MotherAgesBirths* que se describen más adelante (véase Figura 5). Estas clases implementan los método *previous*, *next* y *by_sex* definidos en *DataBase*.

Los principales parámetros de inicialización de la clase abstracta Births son el periodo (*period*), entidades federativa (*state*), sexo (*sex*) y la lista de grupos de edad (*age_groups*). Estos parámetros son almacenados como atributos que permiten identificar el vector de nacimientos. El método *births_filter* realiza consultas a la base de datos para obtener datos de nacimientos con los parámetros indicados. El método abstracto *_init_births_array* tiene el objetivo de inicializar el vector de nacimientos y debe ser implementado por cada subclase de *Births*.

La subclase *StatesBirths* proporciona un vector de nacimientos por entidad federativa, es decir, prepara el vector de nacimientos (atributo *births*) de las 32 entidades federativas del país.

La subclase *MotherAgeBirths* proporciona un vector de nacimientos por edad de la madre en el momento del nacimiento para una sola entidad federativa. Implementa el método *adjust_missing_ages* para aplicar un prorrateo e incorporar los casos de madres sin edad registrada.

1.3.3.2 models.py

Este módulo permite acceder a la base de datos del sistema por medio de las clases *Nacimientos* y *NacimientosEdadMadre*. Además se cuenta con la función *conapo_birth_rates* que proporciona las tasas de natalidad de CONAPO.

1.3.4 Aplicación de Defunciones (*deaths*)

La aplicación *deaths* se encarga de preparar y proporcionar los datos de defunciones y tasas de mortalidad. Esta aplicación tiene un diseño muy similar a la aplicación *births* (véase 1.3.3 Aplicación de Nacimientos (*births*)). Los datos usados fueron obtenidos principalmente de las bases de datos de defunciones generales del Sistema Nacional de Información en Salud de 1979 a 2011. La datos de defunciones se pueden obtener por entidad federativa (entidad de residencia habitual) o por edad.

También se cuenta con las tasas de mortalidad publicadas por el Consejo Nacional de Población de 1990 a 2010.

1.3.4.1 deaths.py

El módulo *deaths* define la clase abstracta *Deaths* que proporciona una estructura general con atributos que ayudan a identificar y almacenar un vector de defunciones. *Deaths*, al igual que *Births*, es una subclase de *DataBase* (véase 1.3.1.1 *base.py*). De la clase *Deaths* se derivan las subclases *StatesDeaths* y *AgeDeaths* que se describen más adelante (véase Figura 6). Estas clases implementan los métodos definidos en *DataBase* y los mismos atributos que *Births* (véase 1.3.3.1 *births.py*).

El método *deaths_filter* realiza consultas a la base de datos para obtener datos de defunciones con los parámetros indicados. El método abstracto *_init_deaths_array* tiene el objetivo de inicializar el vector de defunciones y debe ser implementado por cada subclase de *Deaths*.

La subclase *StatesDeaths* proporciona un vector de defunciones por entidad federativa, es decir, prepara el vector de defunciones (atributo *deaths*) de las 32 entidades federativas del país.

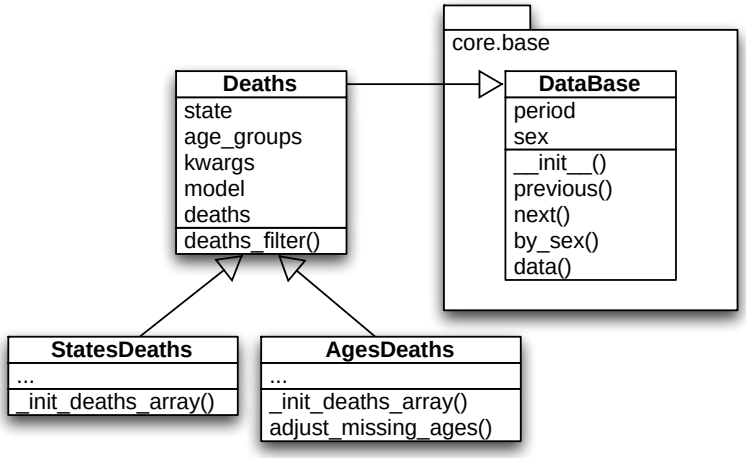

Figura 6. Diagrama de clases: *deaths*

La subclase *AgeDeaths* proporciona un vector de defunciones por edad de la persona al momento de morir para una sola entidad federativa. Implementa el método *adjust_missing_ages* para aplicar un prorrateo e incorporar los casos de defunciones sin edad registrada.

1.3.4.2 models.py

Este módulo permite acceder a la base de datos del sistema por medio de las clases *Defunciones* y se define la función *conapo_death_rates* que proporciona las tasas de mortalidad de CONAPO.

1.3.5 Aplicación de Migración (migration)

La aplicación *migration* se encarga de preparar y proporcionar los datos de migración, incluyendo la migración interna (interestatal) y la migración internacional. Dentro de estos datos, la aplicación puede generar matrices de flujos migratorios internos, es decir, de las entidades federativas de México. Los datos de migración interna son extraídos de las muestras de los datos de censos y conteos, mientras que los de migración internacional solo de las muestras censales.

1.3.5.1 matrix.py

En el módulo *matrix* se define la clase *MigrationMatrix* que proporciona una estructura general con atributos que ayudan a identificar y almacenar una matriz de flujos migratorios interna (véase Figura 7). Los atributos de inicialización e identificación disponibles son el periodo (*period*), sexo (*sex*), rango de edad (*ageRange*) y entidad federativa (*state*). El rango de edad está formado por un

arreglo de números que representan las edades de la población que se toman en cuenta para la construcción de la matriz de flujos migratorios.

Para facilitar su creación y manejo, la matriz de flujos migratorios es generada sobre la estructura de datos *DataFrame* de la biblioteca Pandas. Esta matriz puede ser accedida con el atributo *matrix* de la clase *MigrationMatrix*. Otros atributos que pueden resultar convenientes son los vectores de inmigrantes (*immigrants*), emigrantes (*emigrants*) y saldos netos migratorios (*snm*).

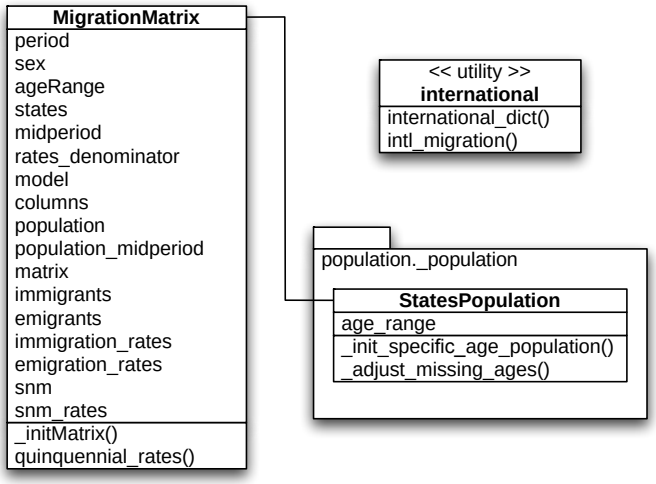

Figura 7. Diagrama de clases: *migration*

MatrixMigration cuenta con los métodos *_initMatrix* y *quinquennial_rates*. El método *_initMatrix* se encarga de inicializar la matriz de flujos migratorios y quinquennial_rates de calcular las tasas de migración con denominador especificado (*rates_denominator*) en el método de inicialización de la clase.

1.3.5.2 international.py

El módulo *international* se encarga de preparar y proporcionar los datos de migración internacional (véase Figura 7). La función *intl_migration* se encarga de proporcionar los vectores de emigrantes e inmigrantes de un periodo específico. El periodo se debe proporcionar como parámetro de la función.

1.3.5.3 models.py

En el módulo *models* se definen los modelos que representan tablas de la base de datos que almacenan los datos condensados de los censos y conteos.

InegiCensoMigracion almacena los registros condensados sobre migración interna. Los datos que se almacenan por registro son el año o periodo en que

realizó el censo o conteo (*censo_ano*), entidad destino (*ent*), entidad de origen o entidad de residencia 5 años antes (*ent_sub5*) y factor de expansión (*factor*).

De forma similar, *InegiCensoMigracionDesplegada* almacena los registros condensados de migración interna, incluyendo en cada registro el sexo y la edad. Haciendo más granular la búsqueda de registros pero también haciéndola más tardada.

La función *conapo_snm* se encarga de generar vectores con los saldos netos migratorios por entidad federativa según las proyecciones de los indicadores demográficos de CONAPO.

1.3.5.4 labels.py

El módulo *labels* incluye funciones auxiliares que proporcionan etiquetas y formatos para la presentación de las matrices de flujos migratorios y algunas otras tablas.

1.3.6 Aplicación de Modelo Matricial (matrixmodel)

En la aplicación *matrixmodel* se implementa el algoritmo para la estimación del saldo neto migratorio con todos los componentes necesarios para su manipulación, ejecución y presentación. Esta aplicación depende de las aplicaciones *core*, *population*, *births*, *deaths* y *migration*, es decir, necesita del código de estas aplicaciones para funcionar correctamente (véase Figura 8).

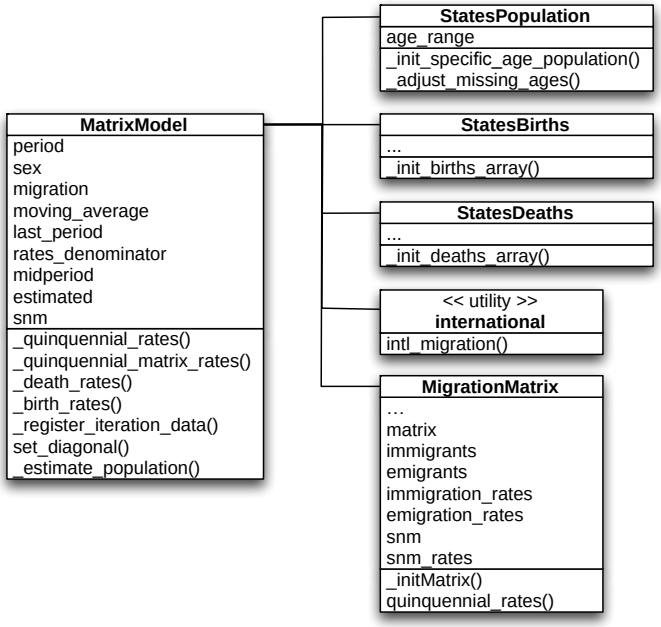

Figura 8. Diagrama de clases: *matrixmodel*

1.3.6.1 matrix_model.py

En el módulo *matrix_model* se define la clase *MatrixModel* (véase Figura 8) que estima la población en base al modelo de distribución territorial de la población (Rogers, 1995). *MatrixModel* también se encarga de implementar el algoritmo para estimarlos saldos netos migratorios (ASNM). Siendo esta una clase medular en la aplicación y proyecto, a continuación se describen sus partes detalladamente.

Para realizar la estimación la clase *MatrixModel* necesita que se establezcan los siguientes parámetros de inicialización (método *__init__*):

- **Periodo inicial (*period*):** periodo del cual se toma la población base y se empiezan a realizar estimaciones.
- **Periodo final (*last_period*):** indica el periodo tope hasta donde se quieren realizar las estimaciones.
- **Bandera "promedios móviles" (*moving_average*):** si es Verdadero, se aplican promedios móviles en el algoritmo para realizar las estimaciones. Si es Falso, se utilizan promedios fijos en el algoritmo. En caso de no proporcionar este parámetro, la bandera tomara el valor de Verdadero.

- **Bandera "migración" (*migration*):** el algoritmo permite habilitar o deshabilitar el uso de los datos de migración interna. Si es Verdadero, se utilizan los datos de migración interna en el algoritmo. Si es Falso, no se incluyen los datos de migración interna. En caso de no proporcionar este parámetro, la bandera tomara el valor de Verdadero.
- **Bandera "migración internacional" (*international*):** el algoritmo también permite habilitar o deshabilitar el uso de datos de migración internacional. Si es Verdadero, se habilitan los datos de migración internacional del periodo indicado (*international_period*). Si es Falso, se ignoran los datos de la migración internacional. En caso de no proporcionar este parámetro, la bandera tomara el valor de Verdadero.
- **Periodo de migración internacional (*international_period*):** indica el periodo de la migración internacional que se desea incluir en el algoritmo. Si la bandera migración internacional no está habilitada (tiene el valor Verdadero), este parámetro es ignorado. Si este parámetro (*international_period*) es proporcionado y la bandera (*international*) está deshabilitada (en Falso), el periodo es ignorado. En el caso de tener la bandera (*international*) habilitada (en Verdadero) y no proporcionar el periodo a utilizar, se deshabilita el uso de la migración internacional automáticamente. Para utilizar la migración internacional es requerido que los dos parámetros (*international*, *international_period*) sean proporcionados.
- **Denominador para calculo de tasas (*rates_denominator*):** el algoritmo requiere utilizar tasas de los componentes demográficos y el denominador para su calculo puede ser modificado con *rates_denominador*. En caso de no proporcionar un valor a este parámetro, se utiliza como denominador 3.5.
- **Bandera "mitad de periodo" (*midperiod*):** indica si se desean utilizar los datos a mitad del periodo, o la mitad del quinquenio. Por ejemplo, si se desean calcular estimaciones de 2005 a 2010, entonces los componentes demográficos se ajustan a la mitad del quinquenio 2005-2010. Esta es una variante del algoritmo que se realizó con fines de investigación.
- **Fuentes alternas de nacimientos y defunciones (*birth_rates_source*, *death_rates_source*):** la clase *MatrixModel* permite alimentar el algoritmo con diferente fuentes de tasas de nacimientos y defunciones. Se tienen que proporcionar vectores con los 32 tasas a utilizar. La aplicación supone que estas tasas han sido multiplicado por 1000 antes de su uso.

Cabe mencionar que en el caso de estar deshabilitado el uso de datos de migración interna e internacional, no se pueden realizar las estimaciones de los saldos netos migratorios. En este caso solo se puede realizar estimaciones de

población (en base a crecimiento natural solamente). El método para estimar saldos netos migratorios requiere que existan datos de migración. En el caso de estar habilitado un solo tipo de migración, ya sea la interna o la internacional, sí se pueden realizar estimaciones de los saldos netos migratorios.

A continuación se describen los métodos más importantes de la clase *MatrixModel* (véase Figura 8). El método *__init__* se encarga de la inicialización y recibe a los parámetros ya descritos. Inicializa algunos componentes necesarios antes de ejecutar el ASNM. Algunos de los componentes que se inicializan en este método son el vector de población, la matriz de flujos migratorios, los vectores de migración internacional. En este mismo método se manda a ejecutar el ASNM (*_estimate_population*) y se crean los vectores que proporcionan las estimaciones de población y SNM.

El método *_estimate_population* se encarga de preparar las tasas de los componentes que se utilizan en el ASNM, incluyendo la matriz de tasas de migración. En este método se utilizan dos matrices, una para la estimación de la población (*matrix_rates*) y otra para la estimación de los SNM (*matrix_rates_snm*). Se manda a ejecutar el método *set_diagonal* que calcula e integra la diagonal para cada una de estas matrices. Una vez listas estas tasas de componentes y las matrices, se entra en un ciclo donde se realizan las estimaciones indicadas empezando con el periodo inicial hasta el periodo final. Es en este ciclo donde se transpone la matriz y se realizan las multiplicaciones de la matriz.

Finalmente, los atributos *estimated* y *snm* son los que proporcionan los resultados de las estimaciones de población y SNM respectivamente. Estos atributos utilizan las estructuras de datos tipo *DataFrame* de *Pandas*.

1.3.6.2 comparison.py

El módulo *comparison* define varias funciones que ayudan a comparar los resultados de las estimaciones con otros datos observados en censos, conteos u otras fuentes.

La función *comparison_stats* permite calcular las diferencias porcentuales entre valores calculados (estimados) y observados. Recibe como parámetros un vector estimado y otro vector estimado, calcula las diferencias porcentuales entre los dos vectores y da como resultado algunos datos estadísticos de estas diferencias. Además de las diferencias porcentuales, se proporcionan como resultados el máximo, mínimo, media, desviación estándar, coeficiente de variación y varianza.

La función *confidence_intervals* calcula el intervalo de confianza de una muestra. En este caso se puede proporcionar como muestra el vector de las estimaciones de una entidad federativa dada.

1.3.6.3 labels.py

El módulo *labels* contiene funciones que preparan los resultados y los etiqueta antes de ser presentados a los usuarios.

1.3.6.4 to_json.py

El módulo *to_json* prepara vectores y otras estructuras de datos convirtiéndolos al tipo JSON para su manipulación por bibliotecas de visualización como NVD3 (véase 1.2.4 Gráficas y visualizaciones: D3.js).

1.3.6.5 views.py

El módulo *views* es responsable de preparar los componentes que forman las vistas que se presentan al usuario. Esto permite tener separada la interfaz de usuario y la presentación de resultados con la lógica del ASNM.

1.4 Condensación y Preparación de datos

INEGI proporciona microdatos que son las bases de datos completas. Estas bases de datos contienen los datos que se obtienen de las encuestas sin un mayor procesamiento. En algunos casos, los archivos de microdatos pueden ser muy grandes. Por ejemplo, se pueden obtener 6.67 GB de los microdatos (sin comprimir) del Censo 2010. En la mayoría de los casos, los microdatos se proporcionan en formato DBF (se describen más adelante) con archivos adicionales que proporcionan los catálogos u otros archivos que nos ayudan a interpretar los datos.

Intentar ejecutar software como la implementación del ASNM, utilizando los microdatos sin procesar, requeriría de tiempos considerables de ejecución. Es por esto que se propone condensar las bases de datos obtenidas de INEGI y otras instituciones antes de ser utilizadas. Esto hace que los tiempos de ejecución disminuyan considerablemente. En este documento, se entiende como condensar datos a obtener un conjunto de muestras representativas de los datos originales. Por ejemplo, para obtener los datos de población por edad desplegada para una entidad especifica, no es necesario acceder a los microdatos de cada censo o conteo. Se pueden extraer los registros del censo o conteo y agrupar por edad, sumando los factores de expansión de los registros (en caso de ser una muestra). Esto permite reducir los registros para una entidad federativa a solo 121 (si se manejan edades de 0 a 120).

1.4.1 Ingreso de otros datos al ASNM

Para ingresar datos diferentes al ASNM se deben fabricar vectores con estructuras de tipo *Series* (de Pandas) o arreglos de *NumPy*. Se recomienda utilizar algún método para condensar la información y evitar el uso de microdatos que ocupan un gran espacio de almacenamiento.

La forma de preparar los datos se deja a su consideración. Un método que se ha utilizado a lo largo del proyecto es leer los archivos .dbf que proporciona INEGI con la biblioteca *dbf* de Python, se procesan y condensan los datos y se almacenan las salidas en un archivo .CSV (*comma separated values*). El código de este proceso puede verse en el archivo *condense.py* del proyecto. Después, este archivo .CSV es cargado a alguna tabla de la base de datos relacional (*load_data.py*), permitiendo el rápido acceso a los datos y que se puedan generar vectores de los mismos. Ejemplos de estos archivos pueden ser revisados en el directorio *data* del proyecto.

1.4.2 DBF (dBASE III)

dBase es el estándar utilizado por INEGI para proporcionar los microdatos de los censos, conteos y encuestas. Los archivos que siguen el formato de este estándar tienen la terminación (extensión) ".dbf". Estos archivos se componen de un encabezado con información sobre la estructura de los datos y los datos mismos. A cada campo de dato se le asigna un tipo de dato (carácter, número, fechas, memo). En el caso de los archivos disponibles en INEGI, los caracteres son guardados con el código Windows ANSI. Es importante mencionar que la descripción de los títulos de cada variable o columna y el significado de los valores en estas bases de datos debe ser consultada en archivos, llamados catálogos, que se proporcionan en un archivo separado. Este catálogo puede ser descargado en archivo Excel, Word o PDF, dependiendo de la base de datos. Por otro lado, INEGI ha adoptado recientemente la Iniciativa de Documentación de Datos (DDI, por sus siglas en inglés) como el estándar para documentar los microdatos de los proyectos de generación de información estadística.

1.4.3 Conclusiones

En este documento se presentaron las partes más importantes del diseño e implementación del sistema ESNM. Se puso particular énfasis en la preparación de datos y la implementación del algoritmo ASNM. Se utilizaron conceptos de programación orientado a objetos con el apoyo de diagramas de clases.

El producto final proporciona un implementación del ASNM con una interfaz de usuario amigable, que visualiza gráficamente los resultados automáticamente y los compara con datos observados en censos o conteos. Esto permite realizar un gran numero de experimentos de forma rápida y confiable.

1.5 Referencias

- Bennett, James, 2008, Practical Django projects, Apress, New York, NY, USA.

- Greenfeld, Daniel, y Roy, Audrey, 2013, Two scoops of django: best practices for Django 1.5, Two Scoops Press, USA.
- Hettinger, Raymond, 2013, Keynote: What makes python awesome? Presentado en PyCon 2013, Santa Clara CA, USA.
- Rogers, Andrei, 1995, Multiregional Demography. Principles, Methods and Extensions, John Wiley & Sons Ltd, West Sussex, England.