```
"""
Tests adding dates to InferenceData
objects with different dim names.

poetry run pytest tests
poetry run pytest tests/test_general_add_dates.py
"""

# %% LIBRARY IMPORTS

from datetime import datetime, timedelta

import arviz as az
import jax.numpy as jnp
import jax.random as jr
import numpy as np
import numpyro
import numpyro.distributions as dist
import polars as pl
import pytest
import forecasttools
import warnings

# suppress warnings
warnings.filterwarnings("ignore")
```

```
# %% MODELS FOR TESTING ADDING DATES


def weekly_rt_nowcast(y=None):
    """Weekly Rt nowcast (random walk)."""
    rt = numpyro.sample("Rt", dist.Normal(1.0, 0.1))
    rt_walk = numpyro.sample("random_walk", dist.Normal(rt, 0.2).expand([12]))
    numpyro.sample("y", dist.Poisson(jnp.exp(rt_walk)), obs=y)


def biweekly_hospitalizations(obs=None):
    """Bi-weekly hospitalizations."""
    lambda_ = numpyro.sample("lambda", dist.Exponential(1.0).expand([10]))
    numpyro.sample("obs", dist.Poisson(lambda_), obs=obs)
```

```python
def daily_hospitalizations(observations=None):
    """Daily hospitalizations."""
    lambda_ = numpyro.sample("lambda", dist.Exponential(1.5).expand([28]))
    numpyro.sample("observations", dist.Poisson(lambda_), obs=observations)
```

```python
# %% MCMC RUNNER AND IDATA GETTER


def run_mcmc(
    model, obs_data, rng_key_int, num_chains, num_samples=500, num_warmup=200
):
    """Gets MCMC object for a given model and observational data"""
    rng_key = jr.key(rng_key_int)
    # set up inference
    kernel = numpyro.infer.NUTS(model)
    mcmc = numpyro.infer.MCMC(
        kernel,
        num_chains=num_chains,
        num_samples=num_samples,
        num_warmup=num_warmup,
    )
    # get model args and used correct arg for
    # provided model
    model_args = model.__code__.co_varnames
    kwargs = {}
    if "y" in model_args:
        kwargs["y"] = obs_data
    if "obs" in model_args:
        kwargs["obs"] = obs_data
    if "observations" in model_args:
        kwargs["observations"] = obs_data
    # run mcmcms
    mcmc.run(rng_key, **kwargs)
    return mcmc


def get_idata_object(model, mcmc, rng_key_int):
    # get posterior samples
    posterior_samples = mcmc.get_samples()
    # get posterior predictive forecast
```

```
    posterior_pred_samples = numpyro.infer.Predictive(
        model, posterior_samples=posterior_samples
    )(rng_key=jr.key(rng_key_int))
    # create InferenceData object
    idata_wo_dates = az.from_numpyro(
        posterior=mcmc, posterior_predictive=posterior_pred_samples
    )
    return idata_wo_dates
```

```
# %% ARTIFICIAL DATA GENERATORS


def generate_weekly_data(rng_key_int: int):
    with numpyro.handlers.seed(rng_seed=rng_key_int):
        return numpyro.sample("weekly_data", dist.Poisson(4.5).expand([12]))


def generate_biweekly_data(rng_key_int: int):
    with numpyro.handlers.seed(rng_seed=rng_key_int):
        return numpyro.sample("biweekly_data", dist.Poisson(6.0).expand([10]))


def generate_daily_data(rng_key_int: int):
    with numpyro.handlers.seed(rng_seed=rng_key_int):
        return numpyro.sample("daily_data", dist.Poisson(8.0).expand([28]))
```

```
# %% GETTING ARTIFICIAL DATA

# generate artificial data
weekly_data = generate_weekly_data(rng_key_int=47)
biweekly_data = generate_biweekly_data(rng_key_int=47)
daily_data = generate_daily_data(rng_key_int=47)


# print example date for rendering sake
print(weekly_data)
print(biweekly_data)
print(daily_data)
```

```
[6 6 4 3 5 4 3 3 2 3 3 1]
[ 2  5  9  3 11  5  4  8  7  6]
```

```
[ 9  9 11 11  6  7  6 10  5  7 10  9 12  6 11  4 10  6  9 14 11  6 11  4
  4  5  4 12]
```

```
# %% RUNNING MODELS, GETTING MCMC OBJECTS

mcmc_weekly = run_mcmc(
    weekly_rt_nowcast, obs_data=weekly_data, rng_key_int=532, num_chains=2
)
mcmc_biweekly = run_mcmc(
    biweekly_hospitalizations,
    obs_data=biweekly_data,
    rng_key_int=532,
    num_chains=3,
)
mcmc_daily = run_mcmc(
    daily_hospitalizations, obs_data=daily_data, rng_key_int=532, num_chains=4
)
```

```
  0%|          | 0/700 [00:00<?, ?it/s]

warmup:   0%|          | 1/700 [00:00<10:20,  1.13it/s, 1 steps of size 2.34e+00. acc. prob=(

sample:  55%|       | 382/700 [00:00<00:00, 529.09it/s, 7 steps of size 5.45e-01. acc. prob=

sample: 100%|     | 700/700 [00:01<00:00, 658.76it/s, 7 steps of size 5.45e-01. acc. prob=0.
```

```
  0%|          | 0/700 [00:00<?, ?it/s]

sample:  54%|       | 376/700 [00:00<00:00, 3753.77it/s, 7 steps of size 4.86e-01. acc. prob

sample: 100%|     | 700/700 [00:00<00:00, 3962.31it/s, 7 steps of size 4.86e-01. acc. prob=0
```

```
  0%|          | 0/700 [00:00<?, ?it/s]
```

```
warmup:   0%|          | 1/700 [00:00<05:38,  2.07it/s, 1 steps of size 2.34e+00. acc. prob=0
sample:  50%|       | 348/700 [00:00<00:00, 792.38it/s, 7 steps of size 6.24e-01. acc. prob=
sample: 100%|     | 700/700 [00:00<00:00, 1056.05it/s, 7 steps of size 6.24e-01. acc. prob=0

  0%|          | 0/700 [00:00<?, ?it/s]
sample:  71%|      | 500/700 [00:00<00:00, 4997.66it/s, 7 steps of size 5.13e-01. acc. prob=
sample: 100%|     | 700/700 [00:00<00:00, 5034.08it/s, 7 steps of size 5.13e-01. acc. prob=0

  0%|          | 0/700 [00:00<?, ?it/s]
sample:  72%|      | 501/700 [00:00<00:00, 5008.89it/s, 7 steps of size 5.24e-01. acc. prob=
sample: 100%|     | 700/700 [00:00<00:00, 5042.49it/s, 7 steps of size 5.24e-01. acc. prob=0

  0%|          | 0/700 [00:00<?, ?it/s]
warmup:   0%|          | 1/700 [00:00<06:06,  1.91it/s, 1 steps of size 2.34e+00. acc. prob=0
sample:  62%|      | 433/700 [00:00<00:00, 926.82it/s, 7 steps of size 5.37e-01. acc. prob=0
sample: 100%|     | 700/700 [00:00<00:00, 1025.87it/s, 7 steps of size 5.37e-01. acc. prob=0

  0%|          | 0/700 [00:00<?, ?it/s]
sample:  61%|       | 427/700 [00:00<00:00, 4261.43it/s, 7 steps of size 4.92e-01. acc. prob
```

```
sample: 100%|     | 700/700 [00:00<00:00, 4336.29it/s, 7 steps of size 4.92e-01. acc. prob=0

  0%|          | 0/700 [00:00<?, ?it/s]

sample:  63%|       | 440/700 [00:00<00:00, 4394.78it/s, 7 steps of size 5.41e-01. acc. prob=

sample: 100%|     | 700/700 [00:00<00:00, 4460.85it/s, 7 steps of size 5.41e-01. acc. prob=0

  0%|          | 0/700 [00:00<?, ?it/s]

sample:  63%|       | 440/700 [00:00<00:00, 4397.18it/s, 7 steps of size 5.14e-01. acc. prob=

sample: 100%|     | 700/700 [00:00<00:00, 4467.16it/s, 7 steps of size 5.14e-01. acc. prob=0
```

```python
# %% CONVERTING TO INFERENCE DATA WITHOUT DATES

idata_weekly = get_idata_object(
    model=weekly_rt_nowcast, mcmc=mcmc_weekly, rng_key_int=67
)
idata_biweekly = get_idata_object(
    model=biweekly_hospitalizations, mcmc=mcmc_biweekly, rng_key_int=67
)
idata_daily = get_idata_object(
    model=daily_hospitalizations, mcmc=mcmc_daily, rng_key_int=67
)

# for rendering's sakes
print(idata_biweekly.posterior_predictive.dims)
print(idata_weekly.posterior_predictive.dims)
print(idata_daily.posterior_predictive.dims)
print(idata_biweekly.posterior_predictive)
print(idata_weekly.posterior_predictive)
print(idata_daily.posterior_predictive)
```

```
FrozenMappingWarningOnValuesAccess({'chain': 3, 'draw': 500, 'obs_dim_0': 10})
FrozenMappingWarningOnValuesAccess({'chain': 2, 'draw': 500, 'y_dim_0': 12})
FrozenMappingWarningOnValuesAccess({'chain': 4, 'draw': 500, 'observations_dim_0': 28})
<xarray.Dataset> Size: 64kB
Dimensions:    (chain: 3, draw: 500, obs_dim_0: 10)
Coordinates:
  * chain      (chain) int64 24B 0 1 2
  * draw       (draw) int64 4kB 0 1 2 3 4 5 6 7 ... 493 494 495 496 497 498 499
  * obs_dim_0  (obs_dim_0) int64 80B 0 1 2 3 4 5 6 7 8 9
Data variables:
    obs        (chain, draw, obs_dim_0) int32 60kB 1 1 14 4 6 1 ... 3 2 4 1 5 0
Attributes:
    created_at:                 2024-10-29T20:34:42.425991+00:00
    arviz_version:              0.20.0
    inference_library:          numpyro
    inference_library_version:  0.15.3
<xarray.Dataset> Size: 52kB
Dimensions:  (chain: 2, draw: 500, y_dim_0: 12)
Coordinates:
  * chain    (chain) int64 16B 0 1
  * draw     (draw) int64 4kB 0 1 2 3 4 5 6 7 ... 493 494 495 496 497 498 499
  * y_dim_0  (y_dim_0) int64 96B 0 1 2 3 4 5 6 7 8 9 10 11
Data variables:
    y        (chain, draw, y_dim_0) int32 48kB 0 1 3 5 1 1 0 5 ... 3 7 3 4 2 3 2
Attributes:
    created_at:                 2024-10-29T20:34:42.070805+00:00
    arviz_version:              0.20.0
    inference_library:          numpyro
    inference_library_version:  0.15.3
<xarray.Dataset> Size: 228kB
Dimensions:             (chain: 4, draw: 500, observations_dim_0: 28)
Coordinates:
  * chain               (chain) int64 32B 0 1 2 3
  * draw                (draw) int64 4kB 0 1 2 3 4 5 ... 494 495 496 497 498 499
  * observations_dim_0  (observations_dim_0) int64 224B 0 1 2 3 ... 24 25 26 27
Data variables:
    observations        (chain, draw, observations_dim_0) int32 224kB 1 4 ... 8
Attributes:
    created_at:                 2024-10-29T20:34:42.852451+00:00
    arviz_version:              0.20.0
    inference_library:          numpyro
    inference_library_version:  0.15.3
```

```python
# %% GET OPTION 3

# copied from test_idata_general_time_representation
# will replace with forecasttools iteration once
# ported over
def option_3_add_dates_as_coords_to_idata(
    idata_wo_dates: az.InferenceData,
    group_date_mapping: dict[str, tuple[str, timedelta, str]],
) -> az.InferenceData:
    """
    Modifies an InferenceData object by
    assigning date arrays to selected
    groups.
    """
    # create initial idata object from received object
    idata_w_dates = idata_wo_dates.copy()

    # iterate over selected groups
    # NOTE: policy not decided for non-selected groups
    for group_name, (
        start_date_iso,
        time_step,
        dim_name,
    ) in group_date_mapping.items():
        # get idata group
        idata_group = getattr(idata_w_dates, group_name, None)
        # skip if group or dim_name is not located
        if idata_group is None:
            print(f"Warning: Group '{group_name}' not found in idata.")
            continue
        if dim_name not in idata_group.dims:
            print(
                f"Warning: Dimension '{dim_name}' not found in group '{group_name}'."
            )
            continue
        # convert start date to a datetime object
        start_date_as_dt = datetime.strptime(start_date_iso, "%Y-%m-%d")
        # get the interval size for this dimension
        interval_size = idata_group.sizes[dim_name]
        # generate date range using the specified group time_step
        # otherwise use 1d; currently not type str.
```

```
        interval_dates = (
            pl.date_range(
                start=start_date_as_dt,
                end=start_date_as_dt + (interval_size - 1) * time_step,
                interval=f"{time_step.days}d" if time_step.days else "1d",
                closed="both",
                eager=True,
            )
            .to_numpy()
            .astype("datetime64[ns]")
        )
        # update coordinates of group for specified dimension
        idata_group_with_dates = idata_group.assign_coords(
            {dim_name: interval_dates}
        )
        # set the modified group back to the idata object
        setattr(idata_w_dates, group_name, idata_group_with_dates)
    return idata_w_dates


# get example inference data
idata_wo_dates = forecasttools.nhsn_flu_forecast_wo_dates
print(idata_wo_dates["observed_data"])
print(idata_wo_dates["posterior_predictive"])


# run example usage of option 3
option_3_idata_w_dates = option_3_add_dates_as_coords_to_idata(
    idata_wo_dates=idata_wo_dates,
    group_date_mapping={
        "observed_data": ("2022-08-08", timedelta(days=1), "obs_dim_0"),
        "posterior_predictive": (
            "2022-08-08",
            timedelta(weeks=1),
            "obs_dim_0",
        ),
    },
)
print(option_3_idata_w_dates["observed_data"])
print(option_3_idata_w_dates["posterior_predictive"])
```

```
<xarray.Dataset> Size: 8kB
```

```
Dimensions:    (obs_dim_0: 488)
Coordinates:
  * obs_dim_0  (obs_dim_0) int64 4kB 0 1 2 3 4 5 6 ... 482 483 484 485 486 487
Data variables:
    obs        (obs_dim_0) int64 4kB ...
Attributes:
    created_at:                 2024-10-09T20:38:19.520548+00:00
    arviz_version:              0.19.0
    inference_library:          numpyro
    inference_library_version:  0.15.3
<xarray.Dataset> Size: 2MB
Dimensions:    (chain: 1, draw: 1000, obs_dim_0: 516)
Coordinates:
  * chain      (chain) int64 8B 0
  * draw       (draw) int64 8kB 0 1 2 3 4 5 6 7 ... 993 994 995 996 997 998 999
  * obs_dim_0  (obs_dim_0) int64 4kB 0 1 2 3 4 5 6 ... 510 511 512 513 514 515
Data variables:
    obs        (chain, draw, obs_dim_0) int32 2MB ...
Attributes:
    created_at:                 2024-10-09T20:38:19.517260+00:00
    arviz_version:              0.19.0
    inference_library:          numpyro
    inference_library_version:  0.15.3
<xarray.Dataset> Size: 8kB
Dimensions:    (obs_dim_0: 488)
Coordinates:
  * obs_dim_0  (obs_dim_0) datetime64[ns] 4kB 2022-08-08 ... 2023-12-08
Data variables:
    obs        (obs_dim_0) int64 4kB ...
Attributes:
    created_at:                 2024-10-09T20:38:19.520548+00:00
    arviz_version:              0.19.0
    inference_library:          numpyro
    inference_library_version:  0.15.3
<xarray.Dataset> Size: 2MB
Dimensions:    (chain: 1, draw: 1000, obs_dim_0: 516)
Coordinates:
  * chain      (chain) int64 8B 0
  * draw       (draw) int64 8kB 0 1 2 3 4 5 6 7 ... 993 994 995 996 997 998 999
  * obs_dim_0  (obs_dim_0) datetime64[ns] 4kB 2022-08-08 ... 2032-06-21
Data variables:
    obs        (chain, draw, obs_dim_0) int32 2MB ...
Attributes:
```

```
  created_at:                 2024-10-09T20:38:19.517260+00:00
  arviz_version:              0.19.0
  inference_library:          numpyro
  inference_library_version:  0.15.3
```

```python
# %% TEST CASES FOR IDATA DATE MAPPINGS


test_cases = [
    (
        idata_weekly,
        {"observed_data": ("2022-08-08", timedelta(weeks=1), "y_dim_0")},
    ),
    (
        idata_biweekly,
        {"observed_data": ("2022-08-08", timedelta(weeks=2), "obs_dim_0")},
    ),
    (
        idata_daily,
        {
            "observed_data": (
                "2022-08-08",
                timedelta(days=1),
                "observations_dim_0",
            )
        },
    ),
]
```

```python
# %% FUNCTION TO PERFORM TEST CASES


# (param1, param2), <list of params as tuples>
@pytest.mark.parametrize("idata, group_date_mapping", test_cases)
def test_option_3_add_dates_as_coords_to_idata(idata, group_date_mapping):
    """
    Tests the option_3_add_dates_as_coords_to_idata function to verify
    that date coordinates are correctly assigned to InferenceData groups
    based on the specified start date, time interval, and dimension name.
    """
    # use option 3 to add dates
```

```
idata_w_dates = option_3_add_dates_as_coords_to_idata(
    idata_wo_dates=idata,
    group_date_mapping=group_date_mapping,  # from test_cases
)
# iterate over selected groups
for group_name, (
    start_date_iso,
    time_step,
    dim_name,
) in group_date_mapping.items():
    # make sure group is in data
    assert hasattr(
        idata_w_dates, group_name
    ), f"Group '{group_name}' not found in idata."
    # get idata group
    idata_group = getattr(idata_w_dates, group_name)
    # make sure dim is in group
    assert (
        dim_name in idata_group.dims
    ), f"Dimension '{dim_name}' not found in group '{group_name}'."
    # convert start date to a datetime object
    start_date_as_dt = datetime.strptime(start_date_iso, "%Y-%m-%d")
    # get the interval size for this dimension
    interval_size = idata_group.sizes[dim_name]
    # generate expected dates based on start_date and time_step
    expected_dates = np.array(
        [
            np.datetime64(start_date_as_dt + i * time_step)
            for i in range(interval_size)
        ]
    )
    # extract resultant dates
    result_dates = idata_group.coords[dim_name].values
    print(expected_dates, result_dates)
    # compare expected dates to actual dates
    # NOTE: need to correct this; not sure whether
    # to retain np.datetime or to use str; not sure
    # how to compare np.datetime array equality (w/
    # tolerance possibly)
    # assert result_dates == expected_dates, (
    #     f"Dates for {group_name} with dimension '{dim_name}' "
```

```
        #     f"do not match expected dates.\nExpected: {expected_dates}\nGot: {result_dat
        # )


# %%
```