Ruby master - Feature #16838

Enumerator::ArithmeticSequence missing allocator for #clone and #dup

05/07/2020 04:12 PM - shan (Shannon Skipper)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

In Ruby 2.5, with an Enumerator:

```
1.step.clone
#=> Enumerator
```

In Ruby 2.6, with an Enumerator::ArithmeticSequence:

```
1.step.clone
#!> TypeError (allocator undefined for Enumerator::ArithmeticSequence)
```

I've gotten around it in 2.6 and 2.7 by checking if an enum is an ArithmeticSequence and reconstituting a new one if so:

```
Range.new(enum.begin, enum.end, enum.exclude_end?) % enum.step
```

Instead of cloning:

```
enum.clone
```

I filed this as a bug rather than feature, since it seemed like a breaking change and I wasn't sure if it was intentional. Thank you!

History

#1 - 05/07/2020 04:33 PM - shan (Shannon Skipper)

After I filed this, al203-cr on #ruby IRC pointed out this seems intentional since rb_undef_alloc_func(rb_cArithSeq); is manually undefined here: https://github.com/ruby/ruby/blob/v2_7_1/enumerator.c#L4068

Treating it as immutable makes sense to me, since an ArithmeticSequence's begin/end/exclude_end?/step can't be mutated without Fiddle. It just surprised me, since they're not frozen.

```
1.step.frozen?
#=> false
```

Should an ArithmeticSequence be frozen or cloneable?

#2 - 05/18/2020 10:44 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN)

- ruby -v deleted (ruby 2.6.6p146 (2020-03-31 revision 67876) [x86_64-darwin19])

- Tracker changed from Bug to Feature

ArithmeticSequence.new and ArithmeticSequence.allocate being undefined was definitely a deliberate change in 25a5227ab188b940d8bbc291bf4c9d62e5d63163.

I'm not sure that breaking clone was expected. In general for frozen objects, I think clone without freeze: false should return self. However, that's currently how Ruby works.