

## schedule\_function

June 19, 2025

We want to save logarithmically, with a arbitrary base.

```
[70]: import numpy as np
import numba
```

```
[39]: base = 1.1
max_index = int(np.log(100)/np.log(base))
index = np.arange(max_index+1)
print(f'{index=}')

```

```
index=array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
              17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
              34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48])

```

```
[40]: base**index[-1]
```

```
[40]: 97.01723378487254
```

```
[41]: step = base**index
print(f'{step=}')2**24
```

```
step=array([ 1.          ,  1.1          ,  1.21          ,  1.331          ,  1.4641          ,
             1.61051          ,  1.771561          ,  1.9487171          ,  2.14358881          ,  2.35794769          ,
             2.59374246          ,  2.85311671          ,  3.13842838          ,  3.45227121          ,  3.79749834          ,
             4.17724817          ,  4.59497299          ,  5.05447028          ,  5.55991731          ,  6.11590904          ,
             6.72749995          ,  7.40024994          ,  8.14027494          ,  8.95430243          ,  9.84973268          ,
             10.83470594          ,  11.91817654          ,  13.10999419          ,  14.42099361          ,  15.86309297          ,
             17.44940227          ,  19.1943425          ,  21.11377675          ,  23.22515442          ,  25.54766986          ,
             28.10243685          ,  30.91268053          ,  34.00394859          ,  37.40434344          ,  41.14477779          ,
             45.25925557          ,  49.78518112          ,  54.76369924          ,  60.24006916          ,  66.26407608          ,
             72.89048369          ,  80.17953205          ,  88.19748526          ,  97.01723378])
step.astype(int)=array([ 1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  3,  3,
                        3,  4,  4,
                        5,  5,  6,  6,  7,  8,  8,  9, 10, 11, 13, 14, 15, 17, 19, 21, 23,
                        25, 28, 30, 34, 37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97])

```

Many indecies 'points' to '1', '2' etc. (for small bases)

The steps we want to save at:

```
[42]: save_steps = set(step.astype(int))
      print(f'{save_steps=}')

```

```
save_steps={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25,
28, 30, 34, 37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97}
```

Challenge: We want to solve the inverse problem:

1. We should save the time step if its in 'save\_steps'
2. The save\_index should be the position of the step in save steps (+1 to make room for step=0)
3. We need to do it on the GPU

```
[71]: @numba.njit()
      def save_flag(step, base):
          if step==1:
              return True
          virtual_index = int(np.log(step+1)/np.log(base))
          virtual_step = int(base**virtual_index)
          return virtual_step==step

```

```
[44]: save_steps_tests = [i for i in range(1,max(save_steps)+1) if save_flag(i,
↪base) ]
      print(save_steps_tests)
      assert(save_steps_tests==list(save_steps))

```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34,
37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97]
```

Success!

The function save\_flag picks the right timestep to save.

More tests:

```
[31]: for base in (1.1, 1.2, 1.3, np.sqrt(2), 2, 3, 4, ):
      max_index = int(np.log(1025)/np.log(base))
      index = np.arange(max_index+1)
      save_steps =sorted(list(set((base**index).astype(int))))
      save_steps_tests = [i for i in range(1,save_steps[-1]+1) if save_flag(i,
↪base) ]
      print(save_steps)
      print(save_steps_tests)
      assert(save_steps_tests==list(save_steps))

```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34,
37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97, 106, 117, 129, 142, 156, 171, 189,
207, 228, 251, 276, 304, 334, 368, 405, 445, 490, 539, 593, 652, 717, 789, 868,
```

955]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34, 37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97, 106, 117, 129, 142, 156, 171, 189, 207, 228, 251, 276, 304, 334, 368, 405, 445, 490, 539, 593, 652, 717, 789, 868, 955]

[1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 18, 22, 26, 31, 38, 46, 55, 66, 79, 95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1020]

[1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 18, 22, 26, 31, 38, 46, 55, 66, 79, 95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1020]

[1, 2, 3, 4, 6, 8, 10, 13, 17, 23, 30, 39, 51, 66, 86, 112, 146, 190, 247, 321, 417, 542, 705, 917]

[1, 2, 3, 4, 6, 8, 10, 13, 17, 23, 30, 39, 51, 66, 86, 112, 146, 190, 247, 321, 417, 542, 705, 917]

[1, 2, 4, 5, 8, 11, 16, 22, 32, 45, 64, 90, 128, 181, 256, 362, 512, 724, 1024]

[1, 2, 4, 5, 8, 11, 16, 22, 32, 45, 64, 90, 128, 181, 256, 362, 512, 724, 1024]

[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

[1, 3, 9, 27, 81, 243, 729]

[1, 3, 9, 27, 81, 243, 729]

[1, 4, 16, 64, 256, 1024]

[1, 4, 16, 64, 256, 1024]

```
[ ]: for base in (1.1, 1.2, 1.3, np.sqrt(2), 2, 3, 4, ):
    max_index = int(np.log(1025)/np.log(base))
    index = np.arange(max_index+1)
    save_steps =sorted(list(set((base**index).astype(int))))
    save_steps_tests = [i for i in range(1,save_steps[-1]+1) if save_flag(i,base) ]
    print(save_steps)
    print(save_steps_tests)
    assert(save_steps_tests==list(save_steps))
```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34, 37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97, 106, 117, 129, 142, 156, 171, 189, 207, 228, 251, 276, 304, 334, 368, 405, 445, 490, 539, 593, 652, 717, 789, 868, 955]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34, 37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97, 106, 117, 129, 142, 156, 171, 189, 207, 228, 251, 276, 304, 334, 368, 405, 445, 490, 539, 593, 652, 717, 789, 868, 955]

[1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 18, 22, 26, 31, 38, 46, 55, 66, 79, 95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1020]

[1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 18, 22, 26, 31, 38, 46, 55, 66, 79, 95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1020]

[1, 2, 3, 4, 6, 8, 10, 13, 17, 23, 30, 39, 51, 66, 86, 112, 146, 190, 247, 321, 417, 542, 705, 917]

[1, 2, 3, 4, 6, 8, 10, 13, 17, 23, 30, 39, 51, 66, 86, 112, 146, 190, 247, 321, 417, 542, 705, 917]

```

[1, 2, 4, 5, 8, 11, 16, 22, 32, 45, 64, 90, 128, 181, 256, 362, 512, 724, 1024]
[1, 2, 4, 5, 8, 11, 16, 22, 32, 45, 64, 90, 128, 181, 256, 362, 512, 724, 1024]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
[1, 3, 9, 27, 81, 243, 729]
[1, 3, 9, 27, 81, 243, 729]
[1, 4, 16, 64, 256, 1024]
[1, 4, 16, 64, 256, 1024]

```

So, we can compute the `save_flag` directly from the step.

Calculating the `save_index` directly is more difficult... So we simply count (but only if we have to save):

```

[73]: @numba.njit()
def schedule(step, base):
    # base should be compiled in on GPU
    if not save_flag(step, base):
        return False, None
    count = 0
    for i in range(step+1):
        if save_flag(i, base):
            count += 1
    return True, count

[ ]: for base in (1.1, 1.2, 1.3, np.sqrt(2), 2, 3, 4, ):
    max_index = int(np.log(1025)/np.log(base))
    index = np.arange(max_index+1)
    save_steps = sorted(list(set((base**index).astype(int))))
    save_steps_tests = [i for i in range(1,save_steps[-1]+1) if save_flag(i,
↪base) ]
    save_index_tests = [schedule(save_step, base)[1] for save_step in
↪save_steps_tests]
    print(f'{base=}')
    print(save_steps)
    print(save_steps_tests)
    print(save_index_tests)
    assert(save_steps_tests==list(save_steps))
    assert(save_index_tests==list(np.arange(1,len(save_steps_tests)+1)))

```

```

base=1.1
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34,
37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97, 106, 117, 129, 142, 156, 171, 189,
207, 228, 251, 276, 304, 334, 368, 405, 445, 490, 539, 593, 652, 717, 789, 868,
955]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 19, 21, 23, 25, 28, 30, 34,
37, 41, 45, 49, 54, 60, 66, 72, 80, 88, 97, 106, 117, 129, 142, 156, 171, 189,
207, 228, 251, 276, 304, 334, 368, 405, 445, 490, 539, 593, 652, 717, 789, 868,
955]

```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57]
```

```
base=1.2
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 18, 22, 26, 31, 38, 46, 55, 66, 79, 95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1020]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 18, 22, 26, 31, 38, 46, 55, 66, 79, 95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1020]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34]
```

```
base=1.3
```

```
[1, 2, 3, 4, 6, 8, 10, 13, 17, 23, 30, 39, 51, 66, 86, 112, 146, 190, 247, 321, 417, 542, 705, 917]
```

```
[1, 2, 3, 4, 6, 8, 10, 13, 17, 23, 30, 39, 51, 66, 86, 112, 146, 190, 247, 321, 417, 542, 705, 917]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

```
base=1.4142135623730951
```

```
[1, 2, 4, 5, 8, 11, 16, 22, 32, 45, 64, 90, 128, 181, 256, 362, 512, 724, 1024]
```

```
[1, 2, 4, 5, 8, 11, 16, 22, 32, 45, 64, 90, 128, 181, 256, 362, 512, 724, 1024]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
base=2
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
base=3
```

```
[1, 3, 9, 27, 81, 243, 729]
```

```
[1, 3, 9, 27, 81, 243, 729]
```

```
[1, 2, 3, 4, 5, 6, 7]
```

```
base=4
```

```
[1, 4, 16, 64, 256, 1024]
```

```
[1, 4, 16, 64, 256, 1024]
```

```
[1, 2, 3, 4, 5, 6]
```

```
[75]: base = 2
step = base**20
print(f'{step=}')
print(schedule(step, base))
%timeit schedule(step, base)
```

```
step=1048576
```

```
(True, 21)
```

```
11.8 ms ± 1.2 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
[ ]:
```