In [1]:
```
import numpy as np
import jax.numpy as jnp
import diffrax
import jax
from jax import grad, jacrev
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy.integrate import solve_ivp
from scipy.integrate import ode
from time import time

jax.config.update("jax_enable_x64", True)
plt.rcParams.update({'font.size':10})
```

In [2]:
```
m = open('ERK_model.net','r')
mLines = m.readlines()

for i in range(len(mLines)):

    if ('begin' in mLines[i])&('parameters' in mLines[i]):
        parInit = i+1

    if ('end' in mLines[i])&('parameters' in mLines[i]):
        parEnd = i

    if ('begin' in mLines[i])&('reactions' in mLines[i]):
        reactInit = i+1

    if ('end' in mLines[i])&('reactions' in mLines[i]):
        reactEnd = i

    if ('begin' in mLines[i])&('species' in mLines[i]):
        speciesInit = i+1

    if ('end' in mLines[i])&('species' in mLines[i]):
        speciesEnd = i

parLines = mLines[parInit:parEnd]
lines = mLines[reactInit:reactEnd]
speciesLines = mLines[speciesInit:speciesEnd]
numPars = len(parLines)
```

In [3]:

```
# parameter block

parListString=''

for parID in range(len(parLines)):

    parLines[parID]=parLines[parID].replace("^","**")

    separated= parLines[parID].split(' ')

    for index in range(len(separated)-1,-1,-1):

        if len(separated[index])==0:
            del separated[index]

    parListString += separated[1]+', '

    print(f'Parameter {parID} named {separated[1]} := {separated[2]}')
    exec(separated[1]+'='+separated[2])


packing = 'par=['+parListString+']'
unpacking = parListString+'=par'

exec(packing)
```

```
Parameter 0 named EGF := 20
Parameter 1 named ERKpp_SOS1_FB := 1.0
Parameter 2 named ERKpp_MEK_FB := 1.0
Parameter 3 named ERKpp_RAF1_FB := 1.0
Parameter 4 named lamb := 1.0
Parameter 5 named RAS_t0_active := 0
Parameter 6 named EGFR_tot := 3e5*lamb
Parameter 7 named RAS_tot := 6e4*lamb
Parameter 8 named SOS_tot := 1e5*lamb
Parameter 9 named RasGAP_tot := 6e3*lamb
Parameter 10 named RAF_tot := 5e5*lamb
Parameter 11 named MEK_tot := 2e5*lamb
Parameter 12 named ERK_tot := 3e6*lamb
Parameter 13 named EKAR3_tot := 1e6*lamb
Parameter 14 named ERKTR_tot := 1e6*lamb
Parameter 15 named a1 := 5e-5*EGF
Parameter 16 named d1 := 1e-2
Parameter 17 named b1 := 1e-5/lamb
Parameter 18 named u1a := 1e-2
Parameter 19 named u1b := 1e+2
Parameter 20 named b2a := 1e-6/lamb
Parameter 21 named u2a := 1
Parameter 22 named b2b := 1e-7/lamb
Parameter 23 named u2b := 1
Parameter 24 named k2a := 1e-4/lamb
Parameter 25 named k2b := 1e-5/lamb
Parameter 26 named b3 := 1e-5/lamb
Parameter 27 named u3 := 1e-2
Parameter 28 named k3 := 1e+2
Parameter 29 named a2 := 1e-7/lamb
Parameter 30 named d2 := 1e-2
Parameter 31 named p1 := 1e-7/lamb
Parameter 32 named q1 := 1e-2
Parameter 33 named p2 := 3e-6/lamb
Parameter 34 named q2 := 1e-2
Parameter 35 named p3 := (3e-9/lamb)*ERKpp_SOS1_FB
Parameter 36 named q3 := 3e-4
Parameter 37 named p4 := (6e-10/lamb)*ERKpp_MEK_FB
Parameter 38 named q4 := 3e-4
Parameter 39 named q5 := 1e+2
Parameter 40 named p6 := (6e-10/lamb)*ERKpp_RAF1_FB
Parameter 41 named q6 := 3e-4
Parameter 42 named a0_ekar3 := 3e-9/lamb
Parameter 43 named d0_ekar3 := 1e-3
Parameter 44 named a0_erktr := 1e-9/lamb
Parameter 45 named d0_erktr := 2e-3
Parameter 46 named _InitialConc1 := RAS_tot*(1-RAS_t0_active)
Parameter 47 named _InitialConc2 := RAS_tot*RAS_t0_active
Parameter 48 named _rateLaw1 := p1*2
Parameter 49 named _rateLaw2 := q1*2
Parameter 50 named _rateLaw3 := p2*2
Parameter 51 named _rateLaw4 := q2*2
Parameter 52 named _rateLaw5 := p3*4
Parameter 53 named _rateLaw6 := p3*3
Parameter 54 named _rateLaw7 := p3*2
Parameter 55 named _rateLaw8 := q3*2
Parameter 56 named _rateLaw9 := q3*3
Parameter 57 named _rateLaw10 := q3*4
Parameter 58 named _rateLaw11 := q5*2
```

In [6]:
```
numSpecies = len(speciesLines)
IC= jnp.zeros((numSpecies,))

for speciesID in range(numSpecies):

    separated= speciesLines[speciesID].split(' ')

    for index in range(len(separated)-1,-1,-1):

        if len(separated[index])==0:
            del separated[index]

    exec('IC=IC.at[speciesID].set('+separated[2]+')')
```

In [7]:
```
reactants = np.zeros((len(lines),10))
products = np.zeros((len(lines),10))
rates = jnp.zeros((len(lines), ))

for reactionID in range(len(lines)):

    separated= lines[reactionID].split(' ')

    for index in range(len(separated)-1,-1,-1):

        if len(separated[index])==0:
            del separated[index]

    reactantSet = separated[1].split(',')
    reactants[reactionID][0]=len(reactantSet)

    for reactantID in range(len(reactantSet)):
        reactants[reactionID][reactantID+1] = int(reactantSet[reactantID])-1

    productSet = separated[2].split(',')
    products[reactionID][0]=len(productSet)

    for productID in range(len(productSet)):
        products[reactionID][productID+1] = int(productSet[productID])-1

    exec( 'rates=rates.at[reactionID].set('+separated[3]+')')
```

In [8]:
```
# dimeriztion needs extra care

numReactions = len(rates)

minNum = 20
dimerizationIndex = np.zeros((numReactions,))

for i in range(len(rates)):

    if minNum > reactants[i][0]:

        minNum = reactants[i][0]

    if (reactants[i][0]==2)&(reactants[i][1]==reactants[i][2]):
        print('reaction ID='+str(i)+' is dimerization!!')
        print(asarray(reactants[i])+1)
        print(asarray(products[i])+1)

        dimerizationIndex[i]=1.

reactants=reactants.astype('int')
products=products.astype('int')
```

In [9]:
```
rates
```

Out[9]:
```
Array([1.0e-03, 1.0e-05, 1.0e-07, 1.0e-02, 1.0e-05, 1.0e+02, 1.0e-02,
       2.0e-07, 1.0e-02, 1.0e-02, 1.0e-06, 1.0e-07, 1.0e-02, 1.0e-07,
       1.0e-02, 1.0e-03, 1.0e-02, 1.0e-02, 1.0e+02, 1.0e-06, 1.0e+00,
       1.0e-07, 1.0e+00, 1.0e-04, 1.0e-05, 2.0e-02, 6.0e-06, 1.0e-03,
       1.0e-03, 1.0e+00, 1.0e+00, 1.0e-04, 1.0e-05, 3.0e-06, 1.0e-02,
       2.0e-02, 1.2e-08, 6.0e-10, 6.0e-10, 6.0e-10, 6.0e-10, 6.0e-10,
       3.0e-09, 1.0e-09, 2.0e-07, 1.0e-07, 2.0e-02, 1.0e-02, 6.0e-06,
       3.0e-06, 9.0e-09, 3.0e-04, 3.0e-04, 3.0e-04, 3.0e-04, 2.0e+02,
       1.0e+02, 3.0e-04, 1.0e-03, 2.0e-03, 6.0e-09, 6.0e-04, 3.0e-09,
       9.0e-04, 1.2e-03], dtype=float64)
```

In [10]:
```
@jax.jit
def RHS(t,X,rates):

    dx = jnp.zeros((numSpecies,))

    for i in range(numReactions):

        flux = rates[i]

        for j in range(1, reactants[i][0]+1):

            flux = flux*X[reactants[i][j]]

        for j in range(1, reactants[i][0]+1):

            dx = dx.at[ reactants[i][j] ].add(-flux)

        for j in range(1, products[i][0]+1):

            dx = dx.at[ products[i][j] ].add(+flux)

    return dx
```

Let's pass 1 parameter ERKpp_SOS1_FB as the free parameter

Note that we are running a very short time, $t \in (0, 20)$**.** </color>

In [27]:

```
tSpan = np.linspace(0,20.0,1001)
tol=1e-6

@jax.jit
def xt(ERKpp_SOS1_FB):

    parListString=''

    for parID in range(len(parLines)):

        if parID!=1: # parID=1 is ERKpp_SOS1_FB

            parLines[parID]=parLines[parID].replace("^","**")

            separated= parLines[parID].split(' ')

            for index in range(len(separated)-1,-1,-1):

                if len(separated[index])==0:
                    del separated[index]

            parListString += separated[1]+', '

            print(f'Parameter {parID} named {separated[1]} := {separated[2]}')
            exec(separated[1]+'='+separated[2])


    packing = 'par=['+parListString+']'
    unpacking = parListString+'=par'

    exec(packing)

    reactants = np.zeros((len(lines),10))
    products = np.zeros((len(lines),10))
    rates = jnp.zeros((len(lines), ))

    for reactionID in range(len(lines)):

        separated= lines[reactionID].split(' ')

        for index in range(len(separated)-1,-1,-1):

            if len(separated[index])==0:
                del separated[index]

        reactantSet = separated[1].split(',')
        reactants[reactionID][0]=len(reactantSet)

        for reactantID in range(len(reactantSet)):
            reactants[reactionID][reactantID+1] = int(reactantSet[reactantID])-
1


        productSet = separated[2].split(',')
        products[reactionID][0]=len(productSet)

        for productID in range(len(productSet)):
            products[reactionID][productID+1] = int(productSet[productID])-1

        exec( 'rates=rates.at[reactionID].set('+separated[3]+')')
```

```
    terms = diffrax.ODETerm(RHS)
    solver = diffrax.Dopri8()
    t0 = tSpan[0]
    t1 = tSpan[-1]
    dt0 = None
    saveat = diffrax.SaveAt(ts=tSpan)
    stepsize_controller = diffrax.PIDController(rtol=tol, atol=tol)
    sol = diffrax.diffeqsolve(terms,solver,t0,t1,dt0,IC,args=(rates),saveat=sav
eat,stepsize_controller=stepsize_controller,max_steps=int(1e12))
    return sol.ys
```

In [28]:
```
ERKpp_SOS1_FB
```

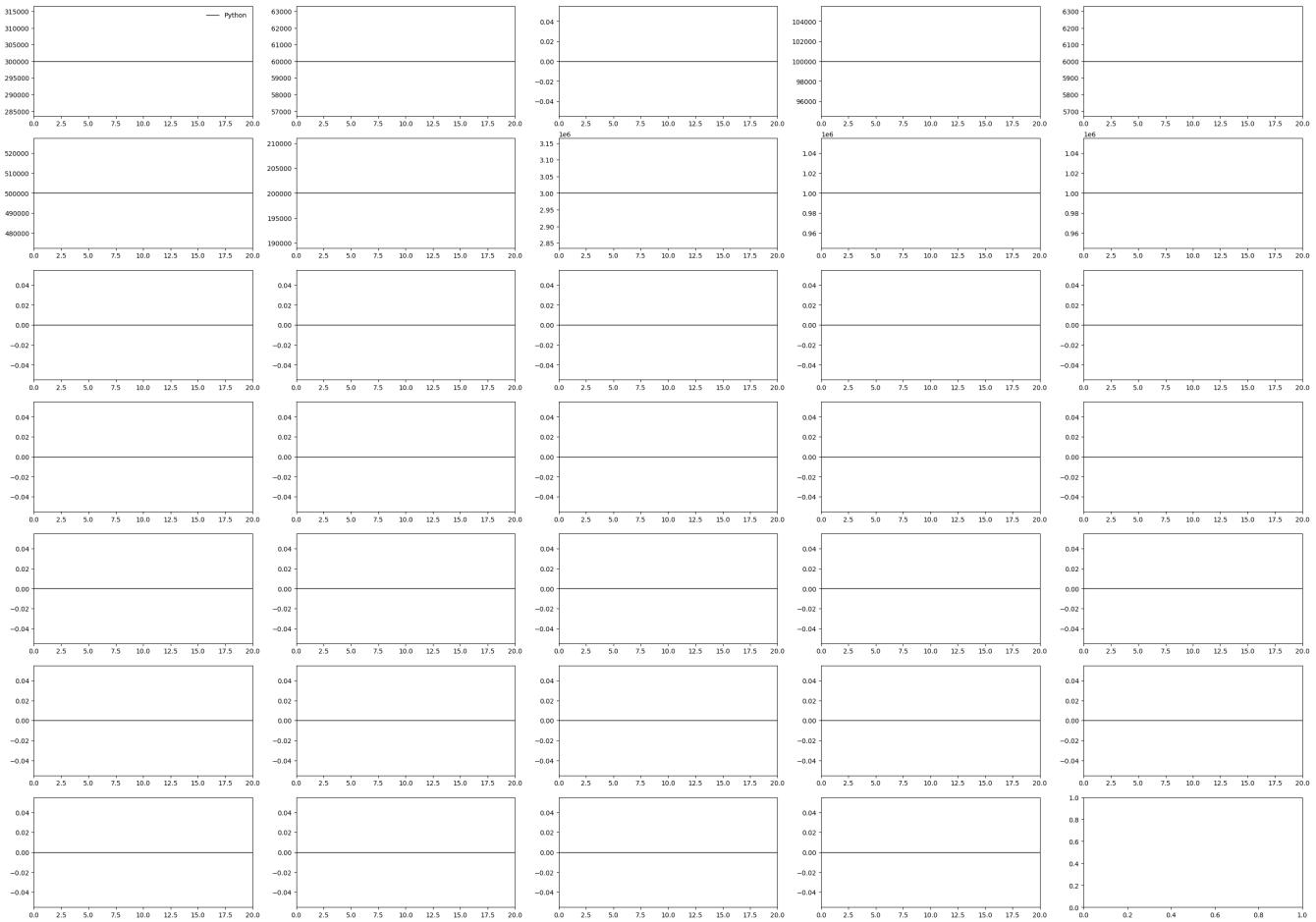
Out[28]: 1.0

In [29]:
```
tic = time()
ys = xt(ERKpp_SOS1_FB).T
print(f'jitting = {time()-tic}')
```

```
Parameter 0 named EGF := 20
Parameter 2 named ERKpp_MEK_FB := 1.0
Parameter 3 named ERKpp_RAF1_FB := 1.0
Parameter 4 named lamb := 1.0
Parameter 5 named RAS_t0_active := 0
Parameter 6 named EGFR_tot := 3e5*lamb
Parameter 7 named RAS_tot := 6e4*lamb
Parameter 8 named SOS_tot := 1e5*lamb
Parameter 9 named RasGAP_tot := 6e3*lamb
Parameter 10 named RAF_tot := 5e5*lamb
Parameter 11 named MEK_tot := 2e5*lamb
Parameter 12 named ERK_tot := 3e6*lamb
Parameter 13 named EKAR3_tot := 1e6*lamb
Parameter 14 named ERKTR_tot := 1e6*lamb
Parameter 15 named a1 := 5e-5*EGF
Parameter 16 named d1 := 1e-2
Parameter 17 named b1 := 1e-5/lamb
Parameter 18 named u1a := 1e-2
Parameter 19 named u1b := 1e+2
Parameter 20 named b2a := 1e-6/lamb
Parameter 21 named u2a := 1
Parameter 22 named b2b := 1e-7/lamb
Parameter 23 named u2b := 1
Parameter 24 named k2a := 1e-4/lamb
Parameter 25 named k2b := 1e-5/lamb
Parameter 26 named b3 := 1e-5/lamb
Parameter 27 named u3 := 1e-2
Parameter 28 named k3 := 1e+2
Parameter 29 named a2 := 1e-7/lamb
Parameter 30 named d2 := 1e-2
Parameter 31 named p1 := 1e-7/lamb
Parameter 32 named q1 := 1e-2
Parameter 33 named p2 := 3e-6/lamb
Parameter 34 named q2 := 1e-2
Parameter 35 named p3 := (3e-9/lamb)*ERKpp_SOS1_FB
Parameter 36 named q3 := 3e-4
Parameter 37 named p4 := (6e-10/lamb)*ERKpp_MEK_FB
Parameter 38 named q4 := 3e-4
Parameter 39 named q5 := 1e+2
Parameter 40 named p6 := (6e-10/lamb)*ERKpp_RAF1_FB
Parameter 41 named q6 := 3e-4
Parameter 42 named a0_ekar3 := 3e-9/lamb
Parameter 43 named d0_ekar3 := 1e-3
Parameter 44 named a0_erktr := 1e-9/lamb
Parameter 45 named d0_erktr := 2e-3
Parameter 46 named _InitialConc1 := RAS_tot*(1-RAS_t0_active)
Parameter 47 named _InitialConc2 := RAS_tot*RAS_t0_active
Parameter 48 named _rateLaw1 := p1*2
Parameter 49 named _rateLaw2 := q1*2
Parameter 50 named _rateLaw3 := p2*2
Parameter 51 named _rateLaw4 := q2*2
Parameter 52 named _rateLaw5 := p3*4
Parameter 53 named _rateLaw6 := p3*3
Parameter 54 named _rateLaw7 := p3*2
Parameter 55 named _rateLaw8 := q3*2
Parameter 56 named _rateLaw9 := q3*3
Parameter 57 named _rateLaw10 := q3*4
Parameter 58 named _rateLaw11 := q5*2
jitting = 3.328986167907715
```

In [30]:
```
fig,ax = plt.subplots(7,5,figsize=(35,25))

for i in range(7):
    for j in range(5):
        index = i*5+j
        if index<34:
            ax[i][j].plot(tSpan, ys[index,:], zorder=2, lw=1, color='k', label
='Python')
            #ax[i][j].plot(bng_t, bng_x[index,:],zorder=1, lw=4, color='y', lab
el='BNG')
            ax[i][j].set_xlim(tSpan[[0,-1]])
            if index==0:
                ax[i][j].legend(loc=1, frameon=False)
```



Apply AD, lo and behold... </color>

In [31]:
```
sensAD = jacrev(xt)(ERKpp_SOS1_FB)
```

```
2023-11-19 21:47:01.157939: W external/tsl/tsl/framework/bfc_allocator.cc:485]
Allocator (GPU_0_bfc) ran out of memory trying to allocate 4.31GiB (rounded to
4633252864)requested by op
2023-11-19 21:47:01.159644: W external/tsl/tsl/framework/bfc_allocator.cc:497]
**********************************************************************
*********_____________
2023-11-19 21:47:01.159682: E external/xla/xla/pjrt/pjrt_stream_executor_clien
t.cc:2716] Execution of replica 0 failed: RESOURCE_EXHAUSTED: Out of memory whi
le trying to allocate 4633252624 bytes.
BufferAssignment OOM Debugging.
BufferAssignment stats:
             parameter allocation:         0B
              constant allocation:         0B
        maybe_live_out allocation:    4.31GiB
     preallocated temp allocation:         0B
                 total allocation:    4.31GiB
              total fragmentation:         0B (0.00%)
Peak buffers:
        Buffer 1:
                Size: 4.31GiB
                Operator: op_name="jit(iota)/jit(main)/iota[dtype=int32 shape=
(34034, 34034) dimension=0]" source_file="/tmp/ipykernel_3923826/758029604.py"
source_line=1
                XLA Label: fusion
                Shape: s32[34034,34034]
                ==========================
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[31], line 1
----> 1 sensAD = jacrev(xt)(ERKpp_SOS1_FB)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/api.py:942, in jacrev.<locals>.jacfun(*args, **kwargs)
    940   y, pullback, aux = _vjp(f_partial, *dyn_args, has_aux=True)
    941 tree_map(partial(_check_output_dtype_jacrev, holomorphic), y)
--> 942 jac = vmap(pullback)(_std_basis(y))
    943 jac = jac[0] if isinstance(argnums, int) else jac
    944 example_args = dyn_args[0] if isinstance(argnums, int) else dyn_args

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/api.py:1031, in _std_basis(pytree)
   1029 ndim = sum(map(np.size, leaves))
   1030 dtype = dtypes.result_type(*leaves)
-> 1031 flat_basis = jnp.eye(ndim, dtype=dtype)
   1032 return _unravel_array_into_pytree(pytree, 1, None, flat_basis)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/numpy/lax_numpy.py:2311, in eye(N, M, k, dtype)
   2309   raise ValueError(f"negative dimensions are not allowed, got {N} and
{M}")
   2310 k = operator.index(k)
-> 2311 return lax_internal._eye(_jnp_dtype(dtype), (N_int, M_int), k)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/lax/lax.py:1247, in _eye(dtype, shape, offset)
   1245 offset = int(offset)
   1246 dtype = dtypes.canonicalize_dtype(dtype)
-> 1247 bool_eye = eq(add(broadcasted_iota(np.int32, shape, 0), np.int32(offse
t)),
   1248               broadcasted_iota(np.int32, shape, 1))
   1249 return convert_element_type_p.bind(bool_eye, new_dtype=dtype, weak_type
=False)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/lax/lax.py:1240, in broadcasted_iota(dtype, shape, dimension)
   1237 static_shape = [None if isinstance(d, core.Tracer) else d for d in shap
e]
   1238 dimension = core.concrete_or_error(
   1239     int, dimension, "dimension argument of lax.broadcasted_iota")
-> 1240 return iota_p.bind(*dynamic_shape, dtype=dtype, shape=tuple(static_shap
e),
   1241                    dimension=dimension)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/core.py:385, in Primitive.bind(self, *args, **params)
    382 def bind(self, *args, **params):
    383   assert (not config.enable_checks.value or
    384           all(isinstance(arg, Tracer) or valid_jaxtype(arg) for arg in
args)), args
--> 385   return self.bind_with_trace(find_top_trace(args), args, params)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_src/core.py:388, in Primitive.bind_with_trace(self, trace, args, params)
    387 def bind_with_trace(self, trace, args, params):
--> 388   out = trace.process_primitive(self, map(trace.full_raise, args), para
```

```
ms)
    389   return map(full_lower, out) if self.multiple_results else full_lower
(out)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_sr
c/core.py:868, in EvalTrace.process_primitive(self, primitive, tracers, params)
    867 def process_primitive(self, primitive, tracers, params):
--> 868   return primitive.impl(*tracers, **params)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_sr
c/dispatch.py:140, in apply_primitive(prim, *args, **params)
    136   msg = pjit._device_assignment_mismatch_error(
    137       prim.name, fails, args, 'jit', arg_names)
    138   raise ValueError(msg) from None
--> 140 return compiled_fun(*args)

File /home/yentingl/anaconda3/envs/AFT_jax/lib/python3.10/site-packages/jax/_sr
c/dispatch.py:172, in xla_primitive_callable.<locals>.<lambda>(*args, **kw)
    170   call = compiled.unsafe_call
    171 if not prim.multiple_results:
--> 172   return lambda *args, **kw: call(*args, **kw)[0]
    173 else:
    174   return call

ValueError: RESOURCE_EXHAUSTED: Out of memory while trying to allocate 46332526
24 bytes.
BufferAssignment OOM Debugging.
BufferAssignment stats:
             parameter allocation:         0B
              constant allocation:         0B
        maybe_live_out allocation:    4.31GiB
     preallocated temp allocation:         0B
                 total allocation:    4.31GiB
              total fragmentation:         0B (0.00%)
Peak buffers:
        Buffer 1:
                Size: 4.31GiB
                Operator: op_name="jit(iota)/jit(main)/iota[dtype=int32 shape=
(34034, 34034) dimension=0]" source_file="/tmp/ipykernel_3923826/758029604.py"
source_line=1
                XLA Label: fusion
                Shape: s32[34034,34034]
                ==========================
```

In [ ]: