MMTP Analysis (Code Doc.)

Evan w/ cr: Anthony and Alexis

HARVARD
UNIVERSITY

Current Progress

1. **Updated** Python2 syntax to Python3
2. Wrote an **explanation** of the batch_local_diff_params.py script
3. **Proposed** a possible way to do nonuniform efficiencies + some analysis **(Slide 31)**
4. **Found** where the z values are inconsistent and will update soon (GeoOctuplet.hh)

Code Explanation (Uniform Efficiencies)

batch_local_diff_params.py

General Overview

1. **Anthony** has code that can run the simulation for set uniform efficiencies.
2. He **loops** over this code
3. In the loop, all pcb's are fixed, then **random entries** can be chosen to be killed [optional]

```
def loop():
        fullEff = 0.1;
        for i in range(10):#np.linspace(0,1,11):
                outDir = "/Users/evancraft" + "/fullEff%.2f"%fullEff
                os.mkdir(outDir)
                main(nKillPCBJobs=2,nKillPCBs=0,fullEff=round(fullEff,2),outDir=outDir)
                fullEff = fullEff + .1;

if __name__ == "__main__":
    loop()
```

Increments efficiencies (fullEff) uniformly from .1 to 1

Calls main function to run simulation for each of these values

loop()

Jobs

Technicalities

1. Each particular simulation is a **“job”**
2. The job is defined by its **efficiency parameters** and the **number of pcb’s** to be killed
3. We can then **execute this job** or rather run the simulation

```
# Input: None
# Output: None
def main(nKillPCBJobs = 1, nKillPCBs = 16, fullEff = 1.0, outDir = "/Users/evancraft/Documents/"):
        jobs = make_jobs(nKillPCBJobs = nKillPCBJobs, nKillPCBs = nKillPCBs, fullEff = fullEff, outDir = outDir)

        # Ru  in parallel
        starttime = time.time()
        pool = multiprocessing.Pool(processes=6) #use all available cores, otherwise specify the number you want as an argument
        for key, vals in jobs.items():
                pool.apply_async(run_job, args=tuple(), kwds=vals)
                time.sleep(3)
        pool.close()
        pool.join()

        print('That took {} seconds'.format(time.time() - starttime))
```

1. main() receives the (efficiency) job information from loop()
2. job is then generated by make_job()

After being generated, the job is run

main()

Generating the Jobs

Generating the Jobs

1. Generates a set of random configurations (**geometry**)
2. For each configuration, **kills** a set number of pcb's
3. Killed pcb's are given the **dead efficiency**
4. Non-killed pcb's are given **fullEff** from the loop

All efficiencies are set to be the fulleff

Information is fed to kill_pcbs()

```
# Input:
#       - Number of different configurations of randomly killed PCBs
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: Dictionary of job args (job_id, job args)
def make_jobs(nKillPCBJobs = 0, # Number of different configurations of randomly killed PCBs
                        nKillPCBs = 0, # Number of PCBs to randomly kill
                        fullEff = 1.0, # Efficiency of full efficienct PCB
                        dead_PCB_eff = 0.0, # Efficiency for killed PCBs
                        outDir = "/Users/evancraft/Documents/"
                        ):
        jobs = {}
        job_full_eff = {'nJobs':1,
                                'outDir': outDir,
                                'nEvents':10**4,
                                'chamber':'large',
                                'bkgRate':0,
                                'efficiencies':formatMMEffString([fullEff]*NPCBS)} #RealEff)}

        # jobs.setdefault('job_full_eff',{}).update(job_full_eff)

        for j in range(nKillPCBJobs):
                job = copy.deepcopy(job_full_eff)
                effs = [float(i.replace("'","")) for i in job['efficiencies'].split(',')]
                job['efficiencies'] = formatMMEffString(kill_pcbs(effs,nKillPCBs,dead_PCB_eff))
                jobs.setdefault('job_{}_PCBs_killed_id_{}'.format(nKillPCBs,j),{}).update(job)

        return jobs
```

make_jobs()

1. Generates a set of random configurations (geometry)
2. For each configuration, kills a set number of pcb's
3. Killed pcb's are given the dead efficiency
4. Non-killed pcb's are given fullEff from the loop

```
# Input:
#       - List of efficiencies
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: list of efficiencies with killed PCBs
def kill_pcbs(effs, nKillPCBs = 0, dead_PCB_eff = 0):
        killed = []
        for i in range(nKillPCBs):
                pcb = int(round(random.uniform(0,len(effs)-1)))
                while pcb in killed:
                        pcb = int(round(random.uniform(0,len(effs)-1)))
                killed.append(pcb)
                effs[pcb] = dead_PCB_eff
        if nKillPCBs == -1:
                effs = []
                for i in range(8):
                        pcb = int(round(random.uniform(0,15)))
                        x = [0]*16
                        x[pcb] = 1
                        effs += x
                print("Number of PCBS, total efficiency: {}, {}".format(len(effs), np.sum(effs)))
        return effs
```

randomly picks a pcb to kill

kill that index in the eff array

(while loop) if pcb is already killed, pick a different one

Returns the updated efficiency array to run_jobs()

kill_pcbs()

Executing the Job

Executing the job

1. We can then call batch_local.py
2. This calls the C++ end of the code to actually run the simulation

```
# Input: job arguments
# Output: None, calls the job operation
def run_job(nJobs = 1,
                        outDir = "/Users/evancraft/Documents/", # Not currently used
                    nEvents = 100,
                    chamber = 'large',
                    bkgRate = 1,
                    efficiencies = [1.0]*NPCBS):
        os.system( " python python/batch_local.py -j {} -a \"-n {} -ch {} -b {} -tree -uvr -e {}\" -o {}".format(nJobs,

        nEvents,

        chamber,

        bkgRate,

        efficiencies,

        outDir))
```

Feeds information to the C++ side of the code to be run

run_job()

Examples

```
Namespace(NET3=False, a="-n 10000 -ch large -b 0 -tree -uvr -e '0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.1,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0'", d=False, j='1', o='/Users/evancraft/fullEff0.10')
```

In this example, all pcb’s are set to .1 and I told it to kill 127 pcb’s

```
SIMULATION SUMMARY:
0 muons triggered out of 8 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

0 uniform eff,
no dead

```
SIMULATION SUMMARY:
74 muons triggered out of 660 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

.3 uniform eff,
no dead

```
SIMULATION SUMMARY:
932 muons triggered out of 3135 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

.5 uniform eff,
no dead

Non-Uniform Efficiencies (Random)

Non-Uniform Efficiencies (Random)

1. We can play around with the killed pcb efficiencies
2. Fix all pcb's to fullEff
3. Randomly choose some of these to be otherEff

Actual Execution

```
# Input:
#       - Number of different configurations of randomly killed PCBs
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: Dictionary of job args (job_id, job args)
def make_jobs(nKillPCBJobs = 0, # Number of different configurations of randomly killed PCBs
                        nKillPCBs = 0, # Number of PCBs to randomly kill
                        fullEff = 1.0, # Efficiency of full efficienct PCB
                        dead_PCB_eff = 0.0, # Efficiency for killed PCBs
                        outDir = "/Users/evancraft/Documents/"
                        ):
        jobs = {}
        job_full_eff = {'nJobs':1,
                                'outDir': outDir,
                                'nEvents':10**4,
                                'chamber':'large',
                                'bkgRate':0,
                                'efficiencies':formatMMEffString([fullEff]*NPCBS)} #RealEff)}

        # jobs.setdefault('job_full_eff',{}).update(job_full_eff)

        for j in range(nKillPCBJobs):
                job = copy.deepcopy(job_full_eff)
                effs = [float(i.replace("'","")) for i in job['efficiencies'].split(',')]
                job['efficiencies'] = formatMMEffString(kill_pcbs(effs,nKillPCBs,dead_PCB_eff))
                jobs.setdefault('job_{}_PCBs_killed_id_{}'.format(nKillPCBs,j),{}).update(job)

        return jobs
```

make_jobs()

1. Set fulleff to be 80% (example)
2. For each configuration, kill a set number of pcb's
3. Set killedEff to be 50% (example)
4. Non-killed pcb's are given the fullEff from the loop

```
# Input: None
# Output: None
def main(nKillPCBJobs = 1, nKillPCBs = 16, fullEff = 1.0, outDir = "/Users/evancraft/Documents/"):
        jobs = make_jobs(nKillPCBJobs = nKillPCBJobs, nKillPCBs = nKillPCBs, fullEff = fullEff, outDir = outDir)

        # Ru  in parallel
        starttime = time.time()
        pool = multiprocessing.Pool(processes=6) #use all available cores, otherwise specify the number you want as an argument
        for key, vals in jobs.items():
                pool.apply_async(run_job, args=tuple(), kwds=vals)
                time.sleep(3)
        pool.close()
        pool.join()

        print('That took {} seconds'.format(time.time() - starttime))
```

We just need to update the killedeff here to be 50%. As of now, it is not set (runs default of 0.0)

After being generated, the job is run and it will emulate being a non-uniform distribution

main()

Real bb5 Efficiencies

Real bb5 Efficiencies (Hack)

1. We could just kill all of the pcb's and set the killed efficiency (individually) to be the real efficiency
2. A bit of a workaround, but straightforward to code

All efficiencies are set to be the fulleff

```
# Input:
#       - Number of different configurations of randomly killed PCBs
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: Dictionary of job args (job_id, job args)
def make_jobs(nKillPCBJobs = 0, # Number of different configurations of randomly killed PCBs
                          nKillPCBs = 0, # Number of PCBs to randomly kill
                          fullEff = 1.0, # Efficiency of full efficienct PCB
                          dead_PCB_eff = 0.0, # Efficiency for killed PCBs
                          outDir = "/Users/evancraft/Documents/"
                          ):
        jobs = {}
        job_full_eff = {'nJobs':1,
                                    'outDir': outDir,
                                    'nEvents':10**4,
                                    'chamber':'large',
                                    'bkgRate':0,
                                    'efficiencies':formatMMEffString([fullEff]*NPCBS)} #RealEff)}

        # jobs.setdefault('job_full_eff',{}).update(job_full_eff)

        for j in range(nKillPCBJobs):
                job = copy.deepcopy(job_full_eff)
                effs = [float(i.replace("'","")) for i in job['efficiencies'].split(',')]
                job['efficiencies'] = formatMMEffString(kill_pcbs(effs,nKillPCBs,dead_PCB_eff))
                jobs.setdefault('job_{}_PCBs_killed_id_{}'.format(nKillPCBs,j),{}).update(job)

        return jobs
```

make_jobs()

1. What we'll do is leave this alone, and just set all the efficiencies as normal
2. Instead, we can kill all the pcb's and when killing them, set them to be the real efficiency

~~randomly picks a pcb to kill~~

```
# Input:
#       - List of efficiencies
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: list of efficiencies with killed PCBs
def kill_pcbs(effs, nKillPCBs = 0, dead_PCB_eff = 0):
        killed = []
        for i in range(nKillPCBs):
                pcb = int(round(random.uniform(0,len(effs)-1)))
                while pcb in killed:
                        pcb = int(round(random.uniform(0,len(effs)-1)))
                killed.append(pcb)
                effs[pcb] = dead_PCB_eff
        if nKillPCBs == -1:
                effs = []
                for i in range(8):
                        pcb = int(round(random.uniform(0,15)))
                        x = [0]*16
                        x[pcb] = 1
                        effs += x
                print("Number of PCBS, total efficiency: {}, {}".format(len(effs), np.sum(effs)))
        return effs
```

Instead of randomly picking a pcb, (for) loop through all of the pcb's and hard set to the realEff

kill_pcbs()

Non-Uniform Efficiencies (Layers)

Non-Uniform Efficiencies (Layers)

1. We can again play around with the killed pcb efficiencies
2. The pcb list is ordered (128 entries), so just choose to kill particular ranges in the list (i.e. 0-7 corresponds to layer 1 and so on)

```
NPCBS = 128 # 16 per layer
RealEff = [0.5695, 0.9597, 0.9868, 0.985, 0.9825, 0.9835, 0.9849, 0.9844,
```

I believe that the pcb's are **ordered,** so if I wanted to mess with layer 1, I would be messing with indices 0-7

So when I'm killing the pcb's on say layer 1, I would just loop over 0-7

loop()

~~randomly picks a pcb to kill~~

```
# Input:
#       - List of efficiencies
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: list of efficiencies with killed PCBs
def kill_pcbs(effs, nKillPCBs = 0, dead_PCB_eff = 0):
        killed = []
        for i in range(nKillPCBs):
                pcb = int(round(random.uniform(0,len(effs)-1)))
                while pcb in killed:
                        pcb = int(round(random.uniform(0,len(effs)-1)))
                killed.append(pcb)
                effs[pcb] = dead_PCB_eff
        if nKillPCBs == -1:
                effs = []
                for i in range(8):
                        pcb = int(round(random.uniform(0,15)))
                        x = [0]*16
                        x[pcb] = 1
                        effs += x
                print("Number of PCBS, total efficiency: {}, {}".format(len(effs), np.sum(effs)))
        return effs
```

Instead of randomly picking a pcb, (for) loop through only indices 0-7

kill_pcbs()

Question from Anthony

For instance, what happens if 1/2 of the PCBs on 2/8 layers have very low efficiency and the rest have around 90%.

```
(base) Evans-MBP-5:oct_sim-master evancraft$ python3 python/batch_layers.py
Namespace(NET3=False, a="-n 10000 -ch large -b 0 -tree -uvr -e '0.9,0.25,0.9,0.2
5,0.25,0.9,0.25,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.25,0.9,0.9,0.9,0.25,0.9,0.25,0.25,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9'", d=False, j='1', o='/Users/evancraft/fullEff0.90')
Output directory :: /Users/evancraft/fullEff0.90/batch-2020-11-17-20h11m12s
```

In this example,
4/8 pcb's on layer 1 set to .25 eff
4/8 pcb's on layer 3 set to .25 eff

Question from Anthony

For instance, what happens if 1/2 of the PCBs on 2/8 layers have very low efficiency and the rest have around 90%.

```
SIMULATION SUMMARY:
8089 muons triggered out of 9040 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Result: 89%

```
9001 muons triggered out of 9582 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Baseline: 94%

Other Examples

```
(base) Evans-MBP-5:oct_sim-master evancraft$ python3 python/batch_layers.py
Namespace(NET3=False, a="-n 10000 -ch large -b 0 -tree -uvr -e '1.0,1.0,1.0,0.2,
1.0,0.2,1.0,0.2,1.0,1.0,1.0,1.0,1.0,1.0,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0'", d=False, j='1', o='/Users/evancraft/fullEff1.00')
Output directory :: /Users/evancraft/fullEff1.00/batch-2020-11-17-19h55m06s
```

In this example,
⅜ pcb’s on layer 1 set to .2 eff
⅞ pcb’s on layer 3 set to .5 eff

Other Examples

```
SIMULATION SUMMARY:
8057 muons triggered out of 8942 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Result: 90%

```
9001 muons triggered out of 9582 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Baseline: 94%

Trigger Project (Code)

Evan w/ cr: Anthony and Alexis

Current Progress

1. **Updated** Python2 syntax to Python3
2. Wrote an **explanation** of the batch_local_diff_params.py script
3. **Proposed** a possible way to do nonuniform efficiencies + some analysis **(Slide 31)**
4. **Found** where the z values are inconsistent and will update soon (GeoOctuplet.hh)

Code Explanation (Uniform Efficiencies)

batch_local_diff_params.py

General Overview

1. **Anthony** has code that can run the simulation for set uniform efficiencies.
2. He **loops** over this code
3. In the loop, all pcb's are fixed, then **random entries** can be chosen to be killed [optional]

```
def loop():
        fullEff = 0.1;
        for i in range(10):#np.linspace(0,1,11):
                outDir = "/Users/evancraft" + "/fullEff%.2f"%fullEff
                os.mkdir(outDir)
                main(nKillPCBJobs=2,nKillPCBs=0,fullEff=round(fullEff,2),outDir=outDir)
                fullEff = fullEff + .1;

if __name__ == "__main__":
    loop()
```

Increments efficiencies (fullEff) uniformly from .1 to 1

Calls main function to run simulation for each of these values

loop()

Jobs

Technicalities

1. Each particular simulation is a **“job”**
2. The job is defined by its **efficiency parameters** and the **number of pcb’s** to be killed
3. We can then **execute this job** or rather run the simulation

```
# Input: None
# Output: None
def main(nKillPCBJobs = 1, nKillPCBs = 16, fullEff = 1.0, outDir = "/Users/evancraft/Documents/"):
        jobs = make_jobs(nKillPCBJobs = nKillPCBJobs, nKillPCBs = nKillPCBs, fullEff = fullEff, outDir = outDir)

        # Ru  in parallel
        starttime = time.time()
        pool = multiprocessing.Pool(processes=6) #use all available cores, otherwise specify the number you want as an argument
        for key, vals in jobs.items():
                pool.apply_async(run_job, args=tuple(), kwds=vals)
                time.sleep(3)
        pool.close()
        pool.join()

        print('That took {} seconds'.format(time.time() - starttime))
```

1. main() receives the (efficiency) job information from loop()
2. job is then generated by make_job()

After being generated, the job is run

main()

Generating the Jobs

Generating the Jobs

1. Generates a set of random configurations (**geometry**)
2. For each configuration, **kills** a set number of pcb's
3. Killed pcb's are given the **dead efficiency**
4. Non-killed pcb's are given **fullEff** from the loop

All efficiencies are set to be the fulleff

Information is fed to kill_pcbs()

```
# Input:
#       - Number of different configurations of randomly killed PCBs
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: Dictionary of job args (job_id, job args)
def make_jobs(nKillPCBJobs = 0, # Number of different configurations of randomly killed PCBs
                          nKillPCBs = 0, # Number of PCBs to randomly kill
                          fullEff = 1.0, # Efficiency of full efficienct PCB
                          dead_PCB_eff = 0.0, # Efficiency for killed PCBs
                          outDir = "/Users/evancraft/Documents/"
                          ):
        jobs = {}
        job_full_eff = {'nJobs':1,
                                    'outDir': outDir,
                                    'nEvents':10**4,
                                    'chamber':'large',
                                    'bkgRate':0,
                                    'efficiencies':formatMMEffString([fullEff]*NPCBS)} #RealEff)}

        # jobs.setdefault('job_full_eff',{}).update(job_full_eff)

        for j in range(nKillPCBJobs):
                job = copy.deepcopy(job_full_eff)
                effs = [float(i.replace("'","")) for i in job['efficiencies'].split(',')]
                job['efficiencies'] = formatMMEffString(kill_pcbs(effs,nKillPCBs,dead_PCB_eff))
                jobs.setdefault('job_{}_PCBs_killed_id_{}'.format(nKillPCBs,j),{}).update(job)

        return jobs
```

make_jobs()

1. Generates a set of random configurations (geometry)
2. For each configuration, kills a set number of pcb's
3. Killed pcb's are given the dead efficiency
4. Non-killed pcb's are given fullEff from the loop

```
# Input:
#       - List of efficiencies
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: list of efficiencies with killed PCBs
def kill_pcbs(effs, nKillPCBs = 0, dead_PCB_eff = 0):
        killed = []
        for i in range(nKillPCBs):
                pcb = int(round(random.uniform(0,len(effs)-1)))
                while pcb in killed:
                        pcb = int(round(random.uniform(0,len(effs)-1)))
                killed.append(pcb)
                effs[pcb] = dead_PCB_eff
        if nKillPCBs == -1:
                effs = []
                for i in range(8):
                        pcb = int(round(random.uniform(0,15)))
                        x = [0]*16
                        x[pcb] = 1
                        effs += x
                print("Number of PCBS, total efficiency: {}, {}".format(len(effs), np.sum(effs)))
        return effs
```

randomly picks a pcb to kill

kill that index in the eff array

(while loop) if pcb is already killed, pick a different one

Returns the updated efficiency array to run_jobs()

kill_pcbs()

Executing the Job

Executing the job

1. We can then call batch_local.py
2. This calls the C++ end of the code to actually run the simulation

```
# Input: job arguments
# Output: None, calls the job operation
def run_job(nJobs = 1,
                        outDir = "/Users/evancraft/Documents/", # Not currently used
                    nEvents = 100,
                    chamber = 'large',
                    bkgRate = 1,
                    efficiencies = [1.0]*NPCBS):
        os.system( " python python/batch_local.py -j {} -a \"-n {} -ch {} -b {} -tree -uvr -e {}\" -o {}".format(nJobs,

        nEvents,

        chamber,

        bkgRate,

        efficiencies,

        outDir))
```

Feeds information to the C++ side of the code to be run

run_job()

Examples

```
Namespace(NET3=False, a="-n 10000 -ch large -b 0 -tree -uvr -e '0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.1,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0'", d=False, j='1', o='/Users/evancraft/fullEff0.10')
```

In this example, all pcb’s are set to .1 and I told it to kill 127 pcb’s

```
SIMULATION SUMMARY:
0 muons triggered out of 8 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

0 uniform eff,
no dead

```
SIMULATION SUMMARY:
74 muons triggered out of 660 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

.3 uniform eff,
no dead

```
SIMULATION SUMMARY:
932 muons triggered out of 3135 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

.5 uniform eff,
no dead

Non-Uniform Efficiencies (Random)

Non-Uniform Efficiencies (Random)

1. We can play around with the killed pcb efficiencies
2. Fix all pcb's to fullEff
3. Randomly choose some of these to be otherEff

Actual Execution

```
# Input:
#       - Number of different configurations of randomly killed PCBs
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: Dictionary of job args (job_id, job args)
def make_jobs(nKillPCBJobs = 0, # Number of different configurations of randomly killed PCBs
                        nKillPCBs = 0, # Number of PCBs to randomly kill
                        fullEff = 1.0, # Efficiency of full efficienct PCB
                        dead_PCB_eff = 0.0, # Efficiency for killed PCBs
                        outDir = "/Users/evancraft/Documents/"
                        ):
        jobs = {}
        job_full_eff = {'nJobs':1,
                                'outDir': outDir,
                                'nEvents':10**4,
                                'chamber':'large',
                                'bkgRate':0,
                                'efficiencies':formatMMEffString([fullEff]*NPCBS)} #RealEff)}

        # jobs.setdefault('job_full_eff',{}).update(job_full_eff)

        for j in range(nKillPCBJobs):
                job = copy.deepcopy(job_full_eff)
                effs = [float(i.replace("'","")) for i in job['efficiencies'].split(',')]
                job['efficiencies'] = formatMMEffString(kill_pcbs(effs,nKillPCBs,dead_PCB_eff))
                jobs.setdefault('job_{}_PCBs_killed_id_{}'.format(nKillPCBs,j),{}).update(job)

        return jobs
```

make_jobs()

1. Set fulleff to be 80% (example)
2. For each configuration, kill a set number of pcb's
3. Set killedEff to be 50% (example)
4. Non-killed pcb's are given the fullEff from the loop

```
# Input: None
# Output: None
def main(nKillPCBJobs = 1, nKillPCBs = 16, fullEff = 1.0, outDir = "/Users/evancraft/Documents/"):
        jobs = make_jobs(nKillPCBJobs = nKillPCBJobs, nKillPCBs = nKillPCBs, fullEff = fullEff, outDir = outDir)

        # Ru  in parallel
        starttime = time.time()
        pool = multiprocessing.Pool(processes=6) #use all available cores, otherwise specify the number you want as an argument
        for key, vals in jobs.items():
                pool.apply_async(run_job, args=tuple(), kwds=vals)
                time.sleep(3)
        pool.close()
        pool.join()

        print('That took {} seconds'.format(time.time() - starttime))
```

We just need to update the killedeff here to be 50%. As of now, it is not set (runs default of 0.0)

After being generated, the job is run and it will emulate being a non-uniform distribution

main()

Real bb5 Efficiencies

Real bb5 Efficiencies (Hack)

1. We could just kill all of the pcb's and set the killed efficiency (individually) to be the real efficiency
2. A bit of a workaround, but straightforward to code

All efficiencies are set to be the fulleff

```
# Input:
#       - Number of different configurations of randomly killed PCBs
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: Dictionary of job args (job_id, job args)
def make_jobs(nKillPCBJobs = 0, # Number of different configurations of randomly killed PCBs
                          nKillPCBs = 0, # Number of PCBs to randomly kill
                          fullEff = 1.0, # Efficiency of full efficienct PCB
                          dead_PCB_eff = 0.0, # Efficiency for killed PCBs
                          outDir = "/Users/evancraft/Documents/"
                          ):
        jobs = {}
        job_full_eff = {'nJobs':1,
                                    'outDir': outDir,
                                    'nEvents':10**4,
                                    'chamber':'large',
                                    'bkgRate':0,
                                    'efficiencies':formatMMEffString([fullEff]*NPCBS)} #RealEff)}

        # jobs.setdefault('job_full_eff',{}).update(job_full_eff)

        for j in range(nKillPCBJobs):
                job = copy.deepcopy(job_full_eff)
                effs = [float(i.replace("'","")) for i in job['efficiencies'].split(',')]
                job['efficiencies'] = formatMMEffString(kill_pcbs(effs,nKillPCBs,dead_PCB_eff))
                jobs.setdefault('job_{}_PCBs_killed_id_{}'.format(nKillPCBs,j),{}).update(job)

        return jobs
```

make_jobs()

1. What we'll do is leave this alone, and just set all the efficiencies as normal
2. Instead, we can kill all the pcb's and when killing them, set them to be the real efficiency

~~randomly picks a pcb to kill~~

```
# Input:
#       - List of efficiencies
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: list of efficiencies with killed PCBs
def kill_pcbs(effs, nKillPCBs = 0, dead_PCB_eff = 0):
        killed = []
        for i in range(nKillPCBs):
                pcb = int(round(random.uniform(0,len(effs)-1)))
                while pcb in killed:
                        pcb = int(round(random.uniform(0,len(effs)-1)))
                killed.append(pcb)
                effs[pcb] = dead_PCB_eff
        if nKillPCBs == -1:
                effs = []
                for i in range(8):
                        pcb = int(round(random.uniform(0,15)))
                        x = [0]*16
                        x[pcb] = 1
                        effs += x
                print("Number of PCBS, total efficiency: {}, {}".format(len(effs), np.sum(effs)))
        return effs
```

Instead of randomly picking a pcb, (for) loop through all of the pcb's and hard set to the realEff

kill_pcbs()

Non-Uniform Efficiencies (Layers)

Non-Uniform Efficiencies (Layers)

1. We can again play around with the killed pcb efficiencies
2. The pcb list is ordered (128 entries), so just choose to kill particular ranges in the list (i.e. 0-7 corresponds to layer 1 and so on)

```
NPCBS = 128 # 16 per layer
RealEff = [0.5695, 0.9597, 0.9868, 0.985, 0.9825, 0.9835, 0.9849, 0.9844,
```

I believe that the pcb's are **ordered,** so if I wanted to mess with layer 1, I would be messing with indices 0-7

So when I'm killing the pcb's on say layer 1, I would just loop over 0-7

loop()

~~randomly picks a pcb to kill~~

```
# Input:
#       - List of efficiencies
#       - Number of PCBs to kill
#       - The dead PCB efficiency
# Output: list of efficiencies with killed PCBs
def kill_pcbs(effs, nKillPCBs = 0, dead_PCB_eff = 0):
        killed = []
        for i in range(nKillPCBs):
                pcb = int(round(random.uniform(0,len(effs)-1)))
                while pcb in killed:
                        pcb = int(round(random.uniform(0,len(effs)-1)))
                killed.append(pcb)
                effs[pcb] = dead_PCB_eff
        if nKillPCBs == -1:
                effs = []
                for i in range(8):
                        pcb = int(round(random.uniform(0,15)))
                        x = [0]*16
                        x[pcb] = 1
                        effs += x
                print("Number of PCBS, total efficiency: {}, {}".format(len(effs), np.sum(effs)))
        return effs
```

Instead of randomly picking a pcb, (for) loop through only indices 0-7

kill_pcbs()

Question from Anthony

For instance, what happens if 1/2 of the PCBs on 2/8 layers have very low efficiency and the rest have around 90%.

```
(base) Evans-MBP-5:oct_sim-master evancraft$ python3 python/batch_layers.py
Namespace(NET3=False, a="-n 10000 -ch large -b 0 -tree -uvr -e '0.9,0.25,0.9,0.2
5,0.25,0.9,0.25,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.25,0.9,0.9,0.9,0.25,0.9,0.25,0.25,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,
0.9,0.9,0.9,0.9,0.9,0.9'", d=False, j='1', o='/Users/evancraft/fullEff0.90')
Output directory :: /Users/evancraft/fullEff0.90/batch-2020-11-17-20h11m12s
```

In this example,

4/8 pcb's on layer 1 set to .25 eff

4/8 pcb's on layer 3 set to .25 eff

Question from Anthony

For instance, what happens if 1/2 of the PCBs on 2/8 layers have very low efficiency and the rest have around 90%.

```
SIMULATION SUMMARY:
8089 muons triggered out of 9040 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Result: 89%

```
9001 muons triggered out of 9582 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Baseline: 94%

Other Examples

```
(base) Evans-MBP-5:oct_sim-master evancraft$ python3 python/batch_layers.py
Namespace(NET3=False, a="-n 10000 -ch large -b 0 -tree -uvr -e '1.0,1.0,1.0,0.2,
1.0,0.2,1.0,0.2,1.0,1.0,1.0,1.0,1.0,1.0,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0'", d=False, j='1', o='/Users/evancraft/fullEff1.00')
Output directory :: /Users/evancraft/fullEff1.00/batch-2020-11-17-19h55m06s
```

In this example,

3/8 pcb's on layer 1 set to .2 eff

7/8 pcb's on layer 3 set to .5 eff

Other Examples

```
SIMULATION SUMMARY:
8057 muons triggered out of 8942 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Result: 90%

```
9001 muons triggered out of 9582 muons that should trigger
0 triggers with spoiled uv hits
n=1: 0 | n=2: 0 | n=3: 0 | n=4: 0
0 bkg uv hits total
0 extra trigger events
0 events where triggers were only made with bkg hits
```

Baseline: 94%