Monte Carlo

Istvan Danko

CLEO 101
June 15, 2007

Outline:
• What is Monte Carlo
• How to generate Monte Carlo
• Accessing Monte Carlo Information:
  DecayTree
  MC truth matching (MC tagging)
What is Monte Carlo?

- Monte Carlo simulation is the systematic use of samples of random numbers to estimate the parameters of an unknown distribution by statistical simulation (random/statistical sampling) where the dimensionality and or complexity of the problem make straightforward numerical solutions impossible or impractical *(Bock and Krischer: The data analysis briefbook)*

![Diagram](image)

- Monte Carlo methods were originally developed to simulate the passage of particles through matter: the first applications was on neutrons by Fermi and later in the Manhattan Project *(Ulam, Neumann, Metropolis)*

- The name refers to the tourist resort in Monaco, famous about its casinos and gambling, because of the involvement of randomness and chance
What is Monte Carlo?

- It is widely used in high-energy physics experiments to design detectors and to simulate how the detector responds to a particular process (interaction, decay etc.)
- For example, if you want to study a certain decay you want to write an analysis code which isolates the interesting events
- MC simulation can help
  - to compare “observables” of your signal and background in order to find the best selection requirements to increase signal significance
  - to determine detector acceptance and selection efficiency for your signal (some of the events are lost due to the finite size of the detector, the inefficiency of the detector, the off-line event reconstruction, or the analysis selection criteria), which you need to take into account when calculating the real production or decay rate
  - To estimate the level of background in the selected events (some other processes can mimic the same experimental signature and can pass your event selection criteria)
What is Monte Carlo?

- At CLEO we readily use two types of Monte Carlo:
  - **Signal MC**: that you generate to simulate one (or a few) specific decay mode(s) which you consider as your signal – mainly used to study acceptance (efficiency) for your process of interest
  - **Generic MC**: that is generated for all data set to simulate all possible (and known) processes and decays – this is mainly used to study background

For CLEO-c data sets there are four types of generic MC:

- DDbar, DDmix, or $\psi(2S)$ depending of beam energy: $e^+e^- \rightarrow c\bar{c}$
- Continuum: $e^+e^- \rightarrow q\bar{q}$ ($q=u,d,s$)
- Radiative return to the $\psi(2S)$: $e^+e^- \rightarrow \psi(2S) \gamma_{ISR}$
- Tau-pair: $e^+e^- \rightarrow \tau^+\tau^-$

**Warning:** MC is simulated data and not real data! It can not be guaranteed that it describes correctly the process you are studying. You have to study carefully the reliability of the simulation by comparing the MC to data, and make correction, or incorporate discrepancies into the systematic error.
Three steps of Monte Carlo simulation

**Event generation**: simulating the event at the physics level – starts with the $e^+e^-$ annihilation, then produces and decays particles according to a user specified decay table and/or a master (generic) decay table until only stable particles remain. The result is the DecayTree.

**Detector simulation (cleog)**: simulating how the detector will respond to the event – particles are propagated through the detector (charged particles bend in the magnetic field) in small steps and at each step the interaction with the detector material, the energy loss and the detector response is simulated; decays of long lived particles (e.g. $K^0_s$) are also simulated in this step; in addition, realistic detector noise (caused by electronics or beam particles) is merged to the simulated event.

**Reconstruction (mcpass2)**: reconstructing tracks and showers from low level detector hits using the the same algorithms that is used for data reconstruction (pass2).
Event generators

- There are several event generators incorporated into the suez framework; these are available as producers (they produce the decay tree):
  - EvtGenProd: used for CLEO-c MC only (it is not available in earlier code releases)
  - BabayagaGeneratorProd: generating QED processes $e^+e^- \rightarrow e^+e^- / \mu^+\mu^- / \gamma\gamma$ (including higher order radiative corrections)
  - BBbarGeneratorProd: similar to EvtGenProd (it has been used mainly for $e^+e^- \rightarrow BB$ simulation at the Y(4S) resonance)
  - QQbarGeneratorProd: to generate continuum $e^+e^- \rightarrow qq$ events (used at higher energy, around the Y resonances)

- Standalone event generator (not incorporated into suez): QQ which provides an interface for several generators: BB, qq, $e^+e^-$, $\mu^+\mu^-$, $\tau^+\tau^-$ (KoralB), etc. Mainly used for CLEO III MC.
  - this is still used for generic $e^+e^- \rightarrow \tau^+\tau^-$ MC production in CLEO-c
  - if you ever need to use this generator you can find an intro from 2003 at http://www.lns.cornell.edu/restricted/CLEO/CLEO3/soft/cleocmc/Tutorials/
EvtGenProd

- For most CLEO-c MC simulation you will use EvtGenProd
  - EvtGenProd assumes that you start with a virtual photon from $e^+e^-$ annihilation ($e^+e^- \rightarrow \gamma^*$) and then you can specify how the $\gamma^*$ (vpho) "decays" using a user decay file
  - It is very flexible: can simulate $D\bar{D}$, continuum (qq), $\psi(2S)$ production etc.

- Documentation:
  http://www.lns.cornell.edu/~ryd/EvtGen/EvtGen_at_CLEO.html

- The master decay file is $C3\_DATA/DECAY.DEC$ (it is a good starting point when you start to write your own decay file)
The user decay file controls what you generate, e.g:

- “myD0” is an alias of D0 (inherits all of its properties) but we can redefine its decay for our need – this is not necessary we could use “D0” as well, but if we want to specify the decay of a particle that can show up in different places in the decay tree then all of these particles will decay according to our new definition, which might not be what you want

- The $D^0$-bar (anti-$D^0$) decay is not specified: it decays generically as it is defined in the master DECAY.DEC file
EvtGen Decay Models

- EvtGen is very flexible to handle different decays and new decay models can be easily added.

- Each decay model is a subroutine that tells EvtGen how to perform the decay taking into account helicity/angular correlations (at the decay amplitude level). For example, in the previous user decay file:
  - **VPHOTOVISR**: decays the virtual photon (vpho) to a vector meson ($\psi''$) and adds an initial state radiation $\gamma$, which in reality is emitted by one of the beam particles.
  - **VSS**: decays a vector meson ($\psi''$) to two (pseudo-)scalar particles ($D^0$).
  - **PHSP**: decays a particle uniformly according to phase space (all final state configurations allowed by energy-momentum conservation are equally likely).

- It is always a good idea to start with the master decay file (DECAY.DEC), find and copy the decay you want to use until you understand this better.

- The EvtGen documentation has a detailed description of all decay models.
**Cleog: detector simulation**

- **In suez** cleog is the command that performs the detector simulation. It also has the capability to perform the event generation beforehand by specifying the event generator.

- In older releases, first you have to load all the necessary producers by running a suez script:

  ```
  run_file $env(C3_SCRIPTS)/cleog_command.tcl
  ```

- Then you can use the cleog command

  - by specifying a valid run number (check in $C3_INFO/data/runinfo.runinfo):
    ```
    cleog gen <generator> <# events> out <output file> run <#> [-start_event <#>] [-nomerge] [-user_decay <file>] [-standard_decay <file>] [-post <command>]
    ```

  - or by specifying a data set (you can use the -job option to advance the run number to something other than the default for that dataset):
    ```
    cleog gen <generator> <# events> out <output file> dataset <#> [-job <#>] [-start_event <#>] [-nomerge] [-user_decay <file>] [-standard_decay <file>] [-post <command>]
    ```
Cleog: detector simulation (2)

- You can also run the detector simulation only by reading the generated events, which has been generated beforehand, from an input file:

  cleog file <input file> out <output file> [-nevents <#>] [-nomerge] [-user_decay <file>] [-standard_decay <file>] [-post <command>]

  - this might be useful if you want to generate CLEO III MC using an old release and use QQ or EvtGen as event generator

- Warning: Detector simulation constants and detector noise is selected based on the run number! So the results slightly depend on the selected run. The best is to sample all runs and data sets (in proportion with their luminosity) when you generate signal MC.

- To learn more type in suez

  cleog help

- Or on the web:

  http://www.lns.cornell.edu/restricted/CLEO/CLEO3/soft/c3mc/howto.html
if [ catch {
    global env
    exception continueEventLoop on

    # Set the output file name and delete any obsolete file
    # with that name
    set fileout $env(OUTDIR)/cg_$env(MCRUN).pds
    if { [ tcl_file exists $fileout ] == 1 } {
        echo "deleting file '$fileout"
        tcl_file delete $fileout
    }

    # Set up the cleog command
    run_file $env(C3_SCRIPTS)/cleog_command.tcl

    cleog gen EvtGenProd $env(NUMEVT) out $fileout
    run $env(MCRUN) -user_decay $env(USERDEC)

    summary
} resultString ] {
    echo "cleog problem ERROR: $resultString"
}
exit

run_cleog.tcl

The environment variables (OUTDIR, MCRUN, NUMEVT, USERDEC) must be set beforehand in the unix shell:
e.g. in bash:
    export MCRUN=205691

in tcsh:
    setenv MCRUN 205691

No line break!
Mcpass2: event reconstruction

- In suez `mcpass2` is the command that performs the reconstruction of tracks, showers etc. from the generated low level hit information.

- In older releases, first you have to load the necessary producers with a `suez` script:

  ```
  run_file $env(C3_SCRIPTS)/mcpass2_command.tcl
  ```

- Then you can use `mcpass2` command: have to specify the input file (cleog output) and the output file:

  ```
  mcpass2 file <cleog output> out <mcpass2 output> [-nevent <#>] [-post <commands>]
  ```
Sample suez scripts to do mcpass2

```tcl
default prompt off
if [ catch {
    global env
    exception continueEventLoop on

    set filein $env(OUTDIR)/cg_$env(MCRUN).pds
    set fileout $env(OUTDIR)/mcp2_$env(MCRUN).pds
    if { [ tcl_file exists $fileout ] == 1 } {
        echo "deleting file "$fileout"
        tcl_file delete $fileout
    }
}

    # Set up the mcpass2 command:
    run_file $env(C3_SCRIPTS)/mcpass2_command.tcl

    mcpass2 file $filein out $fileout

    summary
} catchMessage ] {
    puts "ERROR : runmcpass2 problem: $catchMessage"
}
} exit
```
Software releases

- On the first day you have learnt that there are four types of stable (frozen or fixed) cleo-c software releases: they end with FULL, P2, MCGEN, or MCP2.
  - P2 is used for data reconstruction (you won't use these releases)
  - FULL can be used for general data analysis
  - MCGEN is for generating/simulating Monte Carlo (cleog):
    - The best is to use the most recent MCGEN release so that you use the best and latest simulation code: 20070329_MCGEN (as of today)
  - MCP2 is for reconstructing Monte Carlo (mcpass2):
    - each P2 release has a corresponding MCP2 release with the same date: e.g. 20060117_P2 and 20060117_MCP2
    - the reconstruction code is the same in the corresponding pairs (the only difference is that there are some packages used in one but not in the other)
    - e.g. data37 was reconstructed with 20041104_P2 release, therefore you have to use the corresponding 20041104_MCP2_1 release for the reconstruction of signal MC generated for data37
Software releases (2)

- As of today you can generate MC for the following cleo-c data sets: data31-43

- You have to use the following MCP2 releases for mcpass2:
  - Data31-37 \((\psi(3770))\) and some \(\psi(2S)\) data): 20041104_MCP2_1
  - Data38 (Ds scan data): 20050822_MCP2_1
  - Data39 (DsD*s data at 4170 MeV): 20060117_MCP2
  - Data40-41 (DsD*s data at 4170 MeV): 20060802_MCP2_1
  - Data42 \((\psi(2S)\) data): 20060802_MCP2_1
  - Data43: \((\psi(3770)\) data): 20060802_MCP2_1

- This means you have to change library release between cleog and mcpass2 using the c3rel command
Running the MC generation

- Login to Inx102 or any Linux node of your institution - I assume your shell is bash and the cleo environment is setup correctly
- Copy all the scripts from /nfs/web/public/CLEO/CLEO101/2007/Day6/generate_mc/ to a convenient location of your choice

```bash
export OUTDIR=/cdat/linux/tem/your_user_name/MC/
export MCRUN=205691
export NUMEVT=10
export USERDEC=my_ddbar.dec
c3rel 20070329_MCGEN
suez -f run_cleog.tcl | tee ${OUTDIR}/cg_${MCRUN}.log
c3rel 20041104_MCP2_1
suez -f run_mcpass2.tcl | tee ${OUTDIR}/mcp2_${MCRUN}.log
```

set the environment variables: OUTDIR, MCRUN, NUMEVT, USERDEC
do not forget to create the output dir: /cdat/linux/tem/your_user_name/MC

Only for test – do not generate more than a few events interactively!
Running MC generation in batch mode

- The MC generation is a very CPU and memory-intensive procedure
- Do not generate more than a few events locally on any machine (only to test your scripts)!
- For mass generation you have to submit your job to the batch queue system. The queue will run jobs on the first available node in the computing farm in an order determined by several parameters, most importantly how much CPU time a user has used recently.
- You have to write a short shell script. An example script is provided in ./Day6/generate_mc/run_mc.sh (using bash) and run_mc.csh (using tcsh)
- There is a Solaris and a Linux farm. Linux is much faster than Solaris!
- Cleo-c MC can be generated on both, but CLEO-3 MC only on Solaris (the reconstruction code for the CLEO-3 Si vertex detector was not ported to Linux)!
- You can submit your script to the Linux queue from any Linux node with 
  qsub -l arch=lx24-x86 run_mc.sh
- You must submit your script to the Solaris queue from sol333
  qsub -l gen run_mc.sh
Batch queues

- To check the status of the batch queue use the command 'qstat' which will report about all running (r) and pending (qw/hqw) jobs:

<table>
<thead>
<tr>
<th>job_id</th>
<th>priority</th>
<th>job name</th>
<th>user</th>
<th>status</th>
<th>date/time of submission</th>
<th>farm node running on</th>
</tr>
</thead>
<tbody>
<tr>
<td>403817</td>
<td>0.55661</td>
<td>data_cont_</td>
<td>sdobbs</td>
<td>r</td>
<td>06/12/2006 19:15:06</td>
<td><a href="mailto:all.q@lnx324.lns.cornell.edu">all.q@lnx324.lns.cornell.edu</a></td>
</tr>
<tr>
<td>405055</td>
<td>0.69478</td>
<td>go_1</td>
<td>hajime</td>
<td>r</td>
<td>06/13/2006 11:36:36</td>
<td><a href="mailto:all.q@lnx324.lns.cornell.edu">all.q@lnx324.lns.cornell.edu</a></td>
</tr>
<tr>
<td>403432</td>
<td>0.55016</td>
<td>MCGEN</td>
<td>huanggs</td>
<td>r</td>
<td>06/13/2006 10:50:06</td>
<td><a href="mailto:all.q@lnx728.lns.cornell.edu">all.q@lnx728.lns.cornell.edu</a></td>
</tr>
<tr>
<td>401619</td>
<td>0.55015</td>
<td>mb92921966</td>
<td>jed</td>
<td>r</td>
<td>06/11/2006 20:01:36</td>
<td><a href="mailto:all.q@lnx791.lns.cornell.edu">all.q@lnx791.lns.cornell.edu</a></td>
</tr>
<tr>
<td>401763</td>
<td>0.00000</td>
<td>smc.1-13-6</td>
<td>pcs</td>
<td>hqw</td>
<td>06/09/2006 17:00:41</td>
<td></td>
</tr>
<tr>
<td>401774</td>
<td>0.00000</td>
<td>smc.1-13-5</td>
<td>pcs</td>
<td>hqw</td>
<td>06/09/2006 17:02:10</td>
<td></td>
</tr>
</tbody>
</table>

- If you want to check the status of your own jobs use

  qstat -u user_name

- If you want to run lots of jobs and want to be very polite with other users then you submit subsequent jobs so that each waits for the previous one to finish:

  qsub -hold_jid <job_id> -l <resource name> script2.src

- To delete a job use:  qdel <job_id>

- To learn more about the queue check the man pages for queue, qstat

- The queue should be also used for analyzing the data and MC as well. If you want to run over large amount of data/MC you might need to break up your job to small chunks because running time is limited to 48 hours.
Shell script to run MC in batch mode

• #$ lines are options for the queue engine:
  -S (to specify the shell in which your script will be executed)
  -o (to specify standard output for the shell execution)
  -j y (both standard output and error goes to the same file (specified by -o)

```bash
#!/bin/bash
#$ -S /bin/bash
#$ -o /cdat/linux/tem/idanko/MC/mc_queue.o
#$ -j y
echo Starting
source /nfs/cleo3/Offline/scripts/cleo3logins
source /nfs/cleo3/Offline/scripts/cleo3defs

# Set the run number, number of events to generate, and user decay file:
export NUMEVT=1000
export MCRUN=205691
export USERDEC=my_ddbar.dec
export OUTDIR=/cdat/tem/idanko/CLEO101

etc...
```

run_mc.sh
Looking at the signal MC output

- There are two output files: one from cleog and one from mcpass2, e.g.
  cg_205691.pds (corresponds to the raw data in daq grade of eventstore)
  mcp2_205691.pds (corresponds to the pass2 data in physics grade)
- To analyze the signal MC you usually run on the output from mcpass2.
- You can use any FULL release for analysis (today: 20070330_FULL)
- Since this is your own signal MC which is not available in eventstore you load the data from the file instead of eventstore:
  ```
  source_format sel PDSSourceFormat
  file in /cdat/linux/tem/idanko/CLEO101/mcp2_205691.pds
  setup_analysis
  ```
- Other than that your analysis script should be exactly the same that you use for data analysis.
- For now let's look at the output in EventDisplay: the script is
  `./Day6/scripts/MCinEventDisplay.tcl` (note that both cleog and mcpass2 output is loaded in this case so that we can also look at the detector response but the mcpass2 output is the active source):
  ```
  view -display_only MCGenerator MCDRResponse MCZDResponse MCPass2
  ```
Accessing MC info: MCDecayTree

- **MCDecayTree** holds the information about all the particles and vertexes (interactions) generated in the MC event.

The main components are:

- **MCParticle** class: holds generated particles and provides access to
  - Kinematic variables (derives from KTKinematicData): momentum, energy, etc.
  - Properties (name, charge, flavor, mass etc.)
  - Production and death vertexes
    see MCInfo/MCDecayTree/MCParticle.h

- **MCVertex** class: holds details of interactions
  - Interaction type (decay, scattering, hadronic interaction etc.)
  - Position, time, decay mode
  - Link to parent and children
    see MCInfo/MCDecayTree/MCVertex.h
Accessing the DecayTree

- Extracting the DecayTree in your processor:
  ```
  #include "MCInfo/MCDecayTree/MCDecayTree.h"

  FAItem< MCDecayTree > decayTree;
  //extract( iFrame.record( Stream::kEvent ), decayTree );
  extract( iFrame.record( Stream::kEvent ), decayTree, "Generator" );
  ```

  the first line is to access the DecayTree with everything generated or added by cleog
  the second line with “Generator” usage tag is to access the generator level DecayTree without the additions by cleog

- Iterators for looping over
  - Particles:
    ```
    MCDecayTree::const_pIterator particle = decayTree->pBegin();
    ```
  - Vertexes:
    ```
    MCDecayTree::const_vIterator vertex = decayTree->vBegin();
    ```
Traversing the DecayTree

Find a particular particle in the DecayTree by looping over all the particles:

```cpp
typedef MCDecayTree::const_pIterator MCPartConstIt;

MCPartConstIt treeEnd = decayTree->pEnd();
for( MCPartConstIt partItr = decayTree->pBegin; partItr != treeEnd;
    ++partItr )
{
    const MCParticleProperty& partProp = partItr->properties();

    // Check if this is a Pi0
    if ((partProp.name() == "pi0") )
    {
        report( INFO, kFacilityString )
        << "Particle: " << partProp.name()
        << " p4 = " << partItr->lorentzMomentum() << endl;
    }
}
```

There are more sophisticated functions to do the same job. See `findVertex()`, `findVertexWith()`, `findVertexWithParent()` in MCInfo/MCDecayTree/MCDecayTree.h
MC truth matching (MC tagging)

- MC truth matching is the mapping of the generated MC particles to the reconstructed objects (tracks, showers, etc.):

It is based on how many hits a particle and a reconstructed object share.
Truth matching

- This makes possible to tell (in the MC data) which particle created a particular track

- There might be some complications:
  - More than one track can be matched to a MC particle (e.g. a curler – soft particle that makes one or more full loop inside the drift chamber)
  - More than one MC particle can be matched to a track (e.g. decay in flight – $K \rightarrow \mu \nu$ - with a soft kink)
  - Best matched track for a MC particle might have a different best matched particle (e.g. decay in flight)
  - Showers are even more complicated

- Typically the tagging works quite well.
Standard tagging: Nav object to MCParticle

- Typical example: what is the MCParticle that is best matched to a particular reconstructed object.
- Each Nav object (NavTrack, NavShower, NavKs, NavPi0ToGG) has mcTag() member function which returns a pointer to the best matched MCParticle for that object:

  ```
  const MCParticle& particle = *(track->mcTag());
  report(DEBUG, kFacilityString)
  << "Track " << (*track).identifier() << " is tagged to a "
  << particle.properties().name() << endl;
  ```

- The processor must be linked with MCInfo, MCTrackTagger, and/or MCCCTagger libraries (add them to CLEO3_LIBS list in your Makefile)
- Need to load MCTagHolderProd in your suez script before your processor
- To get access to all matched particles use mcTags() member function.
- Inverse tagging from MCParticle to Nav object is also possible using a list of reconstructed objects indexed by MCParticle: MCTrackTagsByParticle and MCCCTagsByParticle: see tagging documentation at the end of talk.
Further reading

  info on generic MC, generating signal MC, releases, tutorials
- EvtGen documentation: https://www.lns.cornell.edu/%7Eryd/EvtGen/EvtGen_at_CLEO.html
- EvtGen master decay file: $C3\_DATA/DECAY.DEC
- EvtGen particle list: $C3\_DATA/evt.pdl
- Header files: MCDecayTree.h, MCParticle.h, MCVertex.h, MCParticleProperty.h, NavX.h, MCTrackTagsByParticle.h, MCCTagsByParticle.h
Exercises

- Generate a MC sample (1000 events) for the decay chain $e^+e^-\rightarrow\psi(3770)\gamma_{ISR}$, $\psi(3770)\rightarrow D^0D^0$, $D^0\rightarrow K^-\pi^+$ and/or $D^0\rightarrow K^-\pi^+\pi^0$ (the other $D^0$ decays generically): see decay file on page 8, scripts in ./Day6/generate_mc/

- Look at the output in EventDisplay and explore the DecayTree using the script ./Day6/scripts/MCinEventDisplay.tcl

- Run the processor created on Day4 (MyDChainProc) on this signal MC. Compare the plots to those from the data.

- Create a processor DecayTreeProc using
  
  mkproc -mc DecayTreeProc

  and modify it to print out the DecayTree (particles and vertexes), find and count $\pi^0$s, find the $D^0\rightarrow K\pi$ vertex and print the momentum of the $K$ and $\pi$. See ./Day6/src/DecayTreeProc/

- If time permits you can modify MyDChainProc and check truth matching on the children of the $D^0$ candidates that passed the selection (do the $K^-\pi^+$ mode only) to see how many times you got the correct $K^-\pi^+$ combination.