Monte Carlo

Istvan Danko

CLEO 101
June 14, 2006

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 is simulated data.
- It is widely used in high-energy physics experiments to understand how the detector responds to a particular process (interaction, decay etc.)
- A simple example: if you want to measure the rate of a particular process (a decay) you write an analysis code which isolates the interesting events. However, you need to know two things two determine the 'real' rate of the process from the observed number of events:
  - the **efficiency (acceptance)** of the experiment to this type of process; some of the events are lost due to the finite size of the detector, the inefficiency of the detector, the off-line event reconstruction, the analysis selection criteria (cuts), etc.
  - the **level of background** in the selected events; some other processes can mimic the same experimental signature and can pass your event selection criteria
- It is very difficult if not impossible to parametrize the efficiency or the possible background analytically so we have to rely on MC simulation.
What is Monte Carlo?

- **Two types of Monte Carlo:**
  - **Signal MC:** simulating one (or a few) specific decay mode(s) that you consider as your signal – mainly used to study acceptance (efficiency) for your signal process
  - **Generic MC:** when all possible (and known) final states are simulated to reproduce the real data – this is mainly used to study background

**Warning!** MC is 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- interaction, 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. K0s) 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 $e^+e^- \rightarrow e^+e^- / \mu^+\mu^- / \gamma\gamma$ using QED calculations 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) to be used for CLEO III MC: QQ which provides an interface for several generators: BB, qq, mu-pair, Bhabha, tau-pair (KoralB), etc.
  - This is still used for the generic tau-pair MC production
  - 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 DD-bar, 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:

- A new particle “myD0” is defined, which is the same as D0 this way we can redefine its decays.
- Define subsequent decays by specifying the branching fractions, daughters, and decay model.

The $D^0$-bar (anit-D0) 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 (ψ") plus initial state radiation (γ)
  - VSS: decays a vector meson (ψ") to two scalar particles (D^0)
  - PHSP: decays a particle uniformly according to phase space

- 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 details 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.

- First you have to load all the necessary producers etc. in suez:
  
  `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 (this will also generate MC for a single run specified by the -job option):
    
    `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 simulation only using the generated events, which has been generated beforehand, as input from a file (this might be useful if you want to generate CLEO III MC using an old release and use QQ or EvtGen as event generator):

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

- Detector simulation constants and detector noise is selected based on the run number! The best is to sample all runs and data sets 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
Sample suez script to do cleog

```tcl
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
    }
    #load in the cleog command and run cleog
    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 tcsh:
```
setenv MCRUN 205691
```
or in bash:
```
export MCRUN=205691
```
Mcppass2: event reconstruction

- In suez, **mcppass2** is the command that performs the reconstruction of tracks, showers etc. from the low level hit information.

- First you have to load the necessary producers with a suez script:
  
  ```bash
  run_file $env(C3_SCRIPTS)/mcppass2_command.tcl
  ```

- Then you have to specify the input (cleog output) file and the output file:
  
  ```bash
  mcppass2 file <cleog output> out <mcppass2 output> [-nevent <#>] [-post <commands>]
  ```
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 mcpass2 command:
    run_file $env(C3_SCRIPTS)/mcpass2_command.tcl
    mcpass2 file $filein out $fileout
    summary

    } catchMessage ] {
        puts "ERROR : runmcpass2 problem: $catchMessage"
    }

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

- On the first day you have learnt that there are different types of stable (fixed) cleo-c software releases which end with FULL, P2, MGEN, MCP2.
  - P2 is used for data reconstruction (you won't use these releases)
  - FULL can be used for general data analysis
  - MGEN is for generating/simulating Monte Carlo (to do cleog):
    - The best is to use the most recent MGEN release so that you use the best and latest simulation code: 20060426_MGEN (today)
  - MCP2 is for reconstructing Monte Carlo (to do mcpass2):
    - each P2 release has a corresponding MCP2 release (they have the same date), e.g 20060117_P2 and 20060117_MCP2
    - Why?
    - Because you want to use the exact same reconstruction code for the simulated data (MC) which was used for the data reconstruction
    - e.g. since data37 was reconstructed with 20041104_P2 release you have to use the corresponding 20041104_MCP2 release for the reconstruction of signal MC intended to simulate data37 data
Software releases (2)

• As of today you can generate MC for the following cleo-c data sets: data31-39

• You have to use the following MCP2 releases for mcpass2:
  – Data31-37 (psi(3770) and some psi'): 20041104_MCP2_1
  – Data38 (Ds scan data: ): 20050822_MCP2_1
  – Data39 (DsD*s data at 4170 MeV): 20060117_MCP2
  – and so on in the future...

• You have to use two different software releases for cleog and mcpass2! You use the c3rel command to change the release.
Running the MC generation

- Login to lnx102 or any linux node of your institution - I assume your shell is bash (check with 'echo $SHELL') and the cleo environment is setup correctly.

- Copy all the scripts from the examples to a convenient location of your choice

- set the environment variables, do not forget to create the output dir: mkdir /cdat/tem/your_user_name/MC

- Do not generate more than a few events!

```
export OUTDIR=/cdat/tem/your_user_name/MC/
export MCRUN=205691
export NUMEVT=10
export USERDEC=my_ddbar.dec
c3rel 20060426_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
```
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 your job on the first available node in the computing farm in an order determined by several parameters, most importantly how much CPU time you have used recently.
- You have to write a short shell script. An example script is provided in run_mc.scr.
- There is a Solaris and 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 Si vertex detector was not ported to Linux).
- You can submit your script to the Linux queue from any Linux node:
  qsub -l arch=lx24-x86 run_mc.scr
- You must submit your script to the Solaris queue from sol333:
  qsub -l gen run_mc.scr
Batch queues

- To check the status of the 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 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 and run on small chunk because there is a time limit (24 or 48 hours) on how long a job can run.
Shell script to run in batch mode

- #$ lines are additional options for the queue:
  - -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)

```
#!/bin/tcsh
#$ -S /bin/tcsh
#$ -o /cdat/lnsva2/disk3/idanko/mc_test/CLEO101/mc_queue.o
#$ -j y
echo Starting
source /nfs/cleo3/Offline/scripts/cleo3login
source /nfs/cleo3/Offline/scripts/cleo3def
limit datasize unlimited
limit stacksize unlimited

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

etc...
```
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: 20060224_FULL_2)
- Since this is your own signal MC which is not available in eventstore (unless you create your own eventstore database to store your MC) you load the data from the file instead of eventstore in suez:

```tcl
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 would use for data analysis.
- For now let's look at the output in EventDisplay: the script is 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).
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
  - Kinematics (derives from KTKinematicData)
  - 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:

```cpp
#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:
    ```cpp
    MCDecayTree::const_plIterator particle = decayTree->pBegin();
    ```
  - Vertexes:
    ```cpp
    MCDecayTree::const_vlIterator 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 if a particular track (in the MC data) was created by a pion and whether that pion came from a D decay

- There might be some complications:
  - More than one track can be matched to a particle (e.g. a curler – soft particle that makes one or more full loop inside the drift chamber)
  - More than one particle can be matched to a track (e.g. decay in flight – $K \rightarrow \mu \nu$ - with a soft kink)
  - Best matched particle for a track might not be the best matched track for that 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:

```cpp
const MCParticle& particle = *(track->mcTag());
report(DEDEBUG, 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
- MCDecayTree and MC truth matching: Tom Meyer's talk on the 2003 Software workshop page (Day2, Session 5, Accessing Monte Carlo Info):
- MC truth matching web page:
- Header files: MCDecayTree.h, MCParticle.h, MCVertex.h, MCParticleProperty.h, NavX.h, MCTrackTagsByParticle.h, MCCTagsByParticle.h
Exercises

- Generate a MC sample $D^0 \to K^-\pi^+$ and/or $D^0 \to K^-\pi^+\pi^0$ (1000 events)
- Look at the output in EventDisplay, explore the DecayTree
- Run the analysis code created yesterday (MyDChainProc) on this signal MC. Compare the plots to those from the data.
- Run DecayTreeProc on your signal MC and check the output on the screen (or in the log file if you dumped the standard output to a file). Try to modify the code, add more print out or look at other particles.
- 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.