Numerical simulation of gaseous detectors using Garfield++

Supratik Mukhopadhyay
Saha Institute of Nuclear Physics
Workshop on Advanced Radiation Detector and Instrumentation in Nuclear and Particle Physics (RAPID2021)
Outline

• Motivation and background
• Simulation of gaseous ionization detectors
• Introduction to the Garfield++ simulation framework
• Simple Garfield++ how-tos
• Summary and outlook
Disclaimer

• Device physics of particle detectors is a vast subject in itself, touching almost all the aspects of classical physics, glimpses of quantum mechanics and quite some amount of chemistry. As a result, the simulation tools are complex and varied in nature.

• Large number of tool options are available. We will, naturally, touch upon only some of the more common ones, prejudiced by my own exposure to the subject.

• Some of the slides are for later use and will not be discussed in detail during the presentation, to save time.
• Motivation and background
**Ionization detectors**

Detectors that depend on ionization of the media and its registration

**Gas / Liquid Detectors:** Electron - Ion pairs
Geiger-Mueller counter, Proportional counter, Single / Multiple Wire chambers, Drift chambers, Time Projection Chambers, Resistive Plate Chambers, Micro-Pattern Gas Detectors

**Solid State Detectors:** Electron – Hole pairs
Silicon detectors, Diamond detectors

They can both be used as **Tracking detectors** in which you need to know the position to varied degrees of precision.

Here, we are interested in Gaseous ionization detectors

A charged particle passing through a gas-filled counter will ionize the gas along its path. The applied voltage $V$ between the electrodes will attract the positive and negative charges toward the respective electrodes causing a charge $Q$ to be induced on readout electrodes.

The equation governing the ionization process is the well-known Bethe-Bloch formula:

$$-rac{dE}{dx} = K Z^2 \frac{Z}{A} \frac{1}{\beta^2} \left[ \frac{1}{2} \ln \left( \frac{2m_e c^2 \beta^2 \gamma^2 T_{\text{max}}}{I^2} \right) - \beta^2 - \frac{\delta(\beta \gamma)}{2} \right]$$
Operating principle

- Charged particle passes through a volume of a gas or noble liquid exciting and ionizing the molecules: primary electron-ion pairs are created.
- The electrons, if energetic enough through the first interaction, can lead to secondary ionization.
- Electric field is applied by suitably placed electrodes: electrons and ions drift to them
- If the electric field is sufficiently high, the drifting electrons can further ionize the gas (proportional counters)
- At even larger field strengths, electrons emit UV light on the anode (Geiger-Muller counter)
- The movements of electrons and ions induce electronic signal at the readout electrodes. These can be segmented, or otherwise designed to yield information related to position, time etc.
Single Wire Proportional Chamber

Electrons liberated by ionization drift towards the anode wire.
Electrical field close to the wire (typical wire Ø ~few tens of µm) is sufficiently high for electrons (above 10 kV/cm) to gain enough energy to ionize further → avalanche – exponential increase of number of electron ion pairs.

\[ E(r) = \frac{CV_0}{2\pi\varepsilon_0} \cdot \frac{1}{r} \]

- \( C \) – capacitance/unit length

\[ V(r) = \frac{CV_0}{2\pi\varepsilon_0} \cdot \ln \frac{r}{a} \]

Rate capability limited by space charge defined by the time of evacuation of positive ions.

Fortunately, cylindrical geometry is not the only one able to generate strong electric field:
Micro-Pattern Gas Detectors (MPGD)

Printed Circuit Board (PCB) technology allowed micro-structures to be patterned. In the 1990s

- Photolithography
- Etching
- Coating

And later silicon wafer post-processing allowed to go further in small patterns.

Rate Capability Comparison for MWPC and MSGC
RD51 Collaboration

To advance technological development of MPGDs


• ~100 institutes
• ~ 500 people involved
• Representation (Europe, North America, Asia, South America, Africa)

“RD51 aims at facilitating the development of advanced gas-avalanche detector technologies and associated electronic-readout systems, for applications in basic and applied research”

RD51 contributes to the LHC upgrades, BUT, the most important is:

RD51 serves as an access point to MPGD “know-how” for the world-wide community

WG4

Modelling of physics processes and development of relevant software Tools

Trieste, Italy, September 2015
• Simulation of Gaseous Ionization Detectors
Device Physics Simulation

*why is it important, especially for MPGDs*

• **Insight**
  - Optimization of experimental parameters, including environmental ones
  - Accurate interpretation of data
  - Better designs for future detectors

• **Complications**
  - Multiple aspects of complex physics and chemistry
  - Coexistence of large and small length-scales
  - Coexistence of large and small time-scales
Processes occurring in a gaseous detector
Important aspects of a detector

1. Gas mixture
2. Geometry of detector components and the electromagnetic configuration
3. Detector materials and their surface properties (possibility of charging up, roughness, photoionization efficiency)
4. Fabrication expertise, availability of infrastructure
5. Readout, DAQ, Post-processing
6. …
(1) Gas mixture

- Primary production: *easily ionisable*
- Townsend and attachment coefficients: *amplifying, not attaching: doesn't swallow electrons*
- Photon quenching: *preserves spatial localization, avoids streamers / discharges*
- Transport (drift velocity, transverse and longitudinal diffusion): *fast, less diffusing*
- Uncomfortable issues such as flammability, toxicity: *neither flammable, nor explosive, nor toxic*
- Ageing characteristics: *less polymerization, free-flowing*
- Environmentally friendly: *minimizes carbon foot-print*
- Affordability: *keeps costs within limits*
Determines, for a given gas mixture,

- The mode of operation
- Gain
- Resolutions (spatial, temporal, energy)
- Rate handling capability
- Charging up effects
- Space charge effects
- Charge dispersion effects
- Spark probability
- Signal induction
- …
• Introduction to the Garfield++ Simulation Framework
Possible approaches

• Geant4 / FLUKA / SRIM look at the larger picture
  – Important, without doubt, but may be a bit coarse for our purposes.

• Device physics simulation inspects processes occurring at the microscopic detector component level
  – Lumped element (circuit) simulation (Zero dimensional; can extend to realistic problems in combination with other models)
  – Analytic (mostly valid for 1D models)
  – Semi-analytic (1D; certain aspects of more realistic problems)
  – Two-dimensional, axi-symmetric (close to realistic geometries)
  – Three-dimensional (real device, complicated, time consuming)
  – Hydrodynamic (continuum, Euler) model
  – Particle ( discrete, Lagrangian) model
Each and every component is open-source and free


NeBEM is integrated with Garfield++
1. Ionization: PAI and Relaxation (PAIR)

• The transverse distribution of initial ionization aside from the track can be important, but it cannot be obtained from Photo-Absorption and Ionization (PAI). PAI does not discriminate the shells absorbing the energy and does not estimate the range of $\delta$-electrons (ionizing photo-electrons and Auger electrons) knocked out from the atoms.

• The cross section of the PAI model was modified, which resulted in shell separation and gave the possibility to model atomic relaxation cascades and the paths of delta-electrons and fluorescent photons.

➢ The improved model is called Photo-Absorption Ionization and Relaxation (PAIR) model.

First complete, widely available, computer program:

2. Maxwell-Boltzmann Transport

• The MAGBOLTZ program computes drift gas properties by “numerically integrating the Boltzmann transport equation” - i.e., simulating an electron bouncing around inside a gas. By tracking how far the virtual electron propagates, the program can compute the drift velocity. By including a magnetic field, the program can also calculate the Lorentz angle. It can just as easily compute transverse diffusion coefficients, electron mobilities and other parameters.

• In order to find macroscopic parameters like the drift velocity, MAGBOLTZ needs to know about the microscopic nature of each gas under study. The most important quantities are the scattering cross sections, which measure how likely collisions are to occur, and the energy loss per collision.

3. Garfield++
the framework

• Provides a uniform user-interface to all the necessary components.
• Integrates results from the all the components and put them to respective use.
• Does an enormous amount of job in addition - keeps track of primary ionization, drift, diffusion, amplification, loss, signal induced and further post-processing.

4. nearly exact Boundary Element Method (neBEM)

A new formulation based on green’s function that allows the use of exact close-form analytic expressions while solving 3d problems governed by Poisson’s equation. It is very precise even in critical near-field regions, and microscopic length scale.

It is easy to use, interface and integrate neBEM

- Stand-alone
- A driver routine
- An interface routine
- Post-processing

Garfield / Garfield++
Garfield prompt
Garfield script / C++ with Garfield++

Charge density at all the interfaces
Potential at any arbitrary point
Field at any arbitrary point
Capacitance, forces on device components properties can be obtained by post-processing

✓ S Mukhopadhyay and N Majumdar, Computation of 3D MEMS electrostatics using a nearly exact BEM solver EABE 30 (8), 687-696, 2006
• Simple Garfield++ how-tos
Garfield++ webpage

https://garfieldpp.web.cern.ch/garfieldpp/

Garfield++

About

Garfield++ is a toolkit for the detailed simulation of particle detectors based on ionisation measurement in gases and semiconductors. The main area of application is currently in micropattern gaseous detectors. Garfield++ shares functionality with the Garfield program. The main differences are the more up-to-date treatment of electron transport, the possibility to simulate silicon sensors, and the user interface, which is based on ROOT.

More...

Getting started

- Installation
- Examples
- Documentation (User Guide, Doxygen, FAQ)

Support

- If you have any questions, please send a mail to garfield-support@cern.ch (or contact Heinrich Schindler or Rob Veenhof directly).
- To receive (infrequent) announcements about updates of the code, please subscribe to the mailing list garfield-users@cern.ch on E-Groups.
- Issues can be reported on JIRA or GitLab.

Related calculations

- Modelling of avalanches and streamers with COMSOL.
Documentation

User Guide

- Garfield++ User Guide (Version 2021.4)

This document is work in progress. If you find mistakes, inconsistencies, etc. please let me know.

Doxygen

- Doxygen

Frequently Asked Questions

- Questions and Answers
- ROOT forum

Garfield++ User Guide

Version 2021.4

H. Schindler
Prerequisites, installation

Prerequisites

To build the project you need

- a C++ compiler that supports C++11,
- a Fortran compiler,
- ROOT 6,
- GSL (GNU Scientific Library),
- CMake (version 3.9 or higher).

Instructions for installing ROOT can be found here or here.

To get hold of the source files, please follow the steps below.

- Define an environment variable $GARFIELD_HOME pointing to the directory where the Garfield++ source files should be located. Note that this directory must be empty or non-existing.
  If you are using bash, type

  ```bash
  export GARFIELD_HOME=/home/mydir/garfield
  ```

- If you do not have a CERN account, use HTTP authentication.

  ```bash
  git clone https://gitlab.cern.ch/garfield/garfieldpp.git $GARFIELD_HOME
  ```
Get the code from Git
Building the project

- Make sure that the ROOT environment has been set by sourcing the `thisroot.sh` (or `thisroot.csh`) script.
- Go to the directory `$GARFIELD_HOME`.

```
cd $GARFIELD_HOME
```

- Create a build directory and make it your work directory.

```
mkdir build
cd build
```

- Run CMake. If you are happy with the default settings, you can simply type

```
cmake $GARFIELD_HOME
```

- Run make.

```
make -j <number of cores on your machine>
make install
```

- The `install` directory should now contain the headers and the library.
- Set up the environment for building and running a Garfield++-based application by sourcing the script `setupGarfield.(c)sh`. If you are using bash, type

```
source $GARFIELD_HOME/install/share/Garfield/setupGarfield.sh
```

You might want to add these lines to your `.bashrc` (or `.cshrc`). The script will set the environment variable `GARFIELD_INSTALL` pointing to the Garfield++ `install` folder, the environment variable `HEED_DATABASE` pointing to the directory containing the photoabsorption cross-section data, and will preprend the `install` directory to the `CMAKE_PREFIX_PATH` environment variable.

At present, the code is still frequently modified. To obtain the latest version, use the command `git pull origin master` and rebuild the project.
Build Garfield++

```
[supratik@localhost ~]$ echo $GARFIELD_HOME
/home/supratik/AcadPkgs/Garfield++/RAPID2021
[supratik@localhost ~]$ git clone https://gitlab.cern.ch/garfield/garfieldpp.git
$GARFIELD_HOME
Cloning into '/home/supratik/AcadPkgs/Garfield++/RAPID2021'...
remote: Enumerating objects: 14621, done.
remote: Counting objects: 100% (84/84), done.
remote: Compressing objects: 100% (72/72), done.
remote: Total 14621 (delta 39), reused 22 (delta 8), pack-reused 14537
Receiving objects: 100% (14621/14621), 66.52 MiB | 176.00 KiB/s, done.
Resolving deltas: 100% (10751/10751), done.
Updating files: 100% (625/625), done.
[supratik@localhost ~]$  
```
Directory structure
Examples

• Standalone generation of transport properties
  – Magboltz
• Standalone primary generation
  – Heed, Srim, Trim
• Field-solvers
  – Analytic, Ansys, Comsol, CST, Elmer, neBEM
• Gaseous detectors
  – Drift tube, Gem, Micromegas, RPC
• Studies
  – Paschen curve, gain fluctuation, signal generation
• Silicon detectors
  – Standalone, TCAD interfaces
Building and running an application

As an example for building an application, we consider the GEM example located in $GARFIELD_HOME/Examples/Gem.

- Since we will probably want to modify the code, it is a good idea to copy the folder to a local directory.

  ```bash
  cp -r $GARFIELD_HOME/Examples/Gem .
  ```

- The Gem folder contains the source code of the example, the CMakeLists.txt file for building the executable, and the field map files needed for running it.
- Make sure you have set up the ROOT and Garfield++ environment variables by sourcing the scripts thisroot.(c)sh and setupGarfield.(c)sh.
- Create a build folder and make it the current working directory.

  ```bash
  mkdir Gem/build; cd Gem/build
  ```

- Run CMake to create the Makefile and build the executable.

  ```bash
  cmake ..
  make
  ```

- We can now run the application.

  ```bash
  ./gem
  ```
Garfield++ coding

1. Generate transport properties for a chosen gas mixture
2. Create device geometry, assign material properties, voltages on different electrodes
   a) Generate physical field maps
   b) Generate weighting field maps
3. Define sensor related details
4. Define avalanche process
5. Start Event loop
   a) Generate primaries
   b) Transport primaries and their subsequent amplifications till they are attached or goes beyond the gas volume
   c) Compute induced signal
6. Carry out other necessary analysis

Plots can be generated on the fly using ROOT and other interfaces.
Check out the first step
transport properties

- Analyse related example codes
- Build them
- Execute one that generates a file containing transport properties in a gas mixture

```cpp
#include <iostream>
#include "Garfield/MediumMagboltz.hh"
#include "Garfield/FundamentalConstants.hh"

using namespace Garfield;

int main(int argc, char * argv[]) {
    const double pressure = 3 * AtmosphericPressure;
    const double temperature = 293.15;

    // Setup the gas.
    MediumMagboltz gas;
    gas.SetComposition("Ar", 93., "CO2", 7.);
    gas.SetTemperature(temperature);
    gas.SetPressure(pressure);

    // Set field range to be covered by gas table
    const size_t nE = 20;
    const double emin = 100.;
    const double emax = 100000.;

    // Flag to request logarithmic spacing.
    constexpr bool useLog = true;
    gas.SetFieldGrid(emin, emax, nE, useLog);

    const int ncoll = 10;
    // Run Magboltz to generate the gas table.
    gas.GenerateGasTable(ncoll);
    // Save the table.
    gas.WriteGasFile("ar_93_co2_7.gas");
}
```
Build and execute

```
[supratik@localhost ~]$ echo $GARFIELD_HOME
/home/supratik/AcadPkgs/Garfield++/RAPID2021
[supratik@localhost ~]$ git clone https://gitlab.cern.ch/garfield/garfieldpp.git $GARFIELD_HOME
Cloning into '/home/supratik/AcadPkgs/Garfield++/RAPID2021'...
remote: Enumerating objects: 14621, done.
remote: Counting objects: 100% (84/84), done.
remote: Compressing objects: 100% (72/72), done.
remote: Total 14621 (delta 39), reused 22 (delta 8), pack-reused 14537
Receiving objects: 100% (14621/14621), 66.52 MiB | 176.00 KiB/s, done.
Resolving deltas: 100% (10751/10751), done.
Updating files: 100% (625/625), done.
[supratik@localhost ~]$ 
```
1. Transport properties for Ar-CO$_2$ (93:7)
2. Micromegas geometry, assign material properties, voltages on different electrodes
   a) physical field maps using neBEM
   b) weighting field maps using neBEM
3. One continuous anode for signal
4. Monte-Carlo avalanche process
5. Muon event loop
   a) primaries due to an one GeV muon
   b) Transport charges
   c) Induced signal on the anode
6. Check results
// Define materials

// Conducting medium
MediumConductor Cu;

// Gas mixture
MediumMagboltz *gas = new MediumMagboltz();
const double pressure = 760.;
const double temperature = 293.15;
gas->SetTemperature(temperature);
gas->SetPressure(pressure);
gas->SetComposition("Ar", 93., "CO2", 7.);
gas->LoadGasFile("../93Ar_7CO2.gas");
The Micro-mesh Gaseous Structure (Micromegas) is a popular MPGD.

Invented in early 1990s by Ioannis Giomataris, George Charpak et al.

Successfully employed in various experiments including COMPASS, T2K and proposed in several future ones, such as ATLAS.

Excellent position and energy resolution.

Can be made extremely fast (less than 25 ps has been achieved by the PICOSEC collaboration).

High efficiency, low ion back-flow.

Very low material budget.

Can be discharge prone.

Specifications

- Active area: 10x10 cm² to several meters
- Amplification gap: 64 /128 / 192 / 220 µm
- SS wire diameter: 18 µm, pitch 63 / 78 µm
- Spacer diameter: 400 µm, pitch 2 mm
Numerical Bulk Micromegas

Model using Garfield-neBEM
A drift plane
One micromesh, or two, or more
Anode, continuous, or strips
A dielectric substrate
Spacer(s), if needed
Voltages
Device geometry, material properties, voltage configuration

// Geometry.
GeometrySimple geo;
geo.SetMedium(gas);

// Dimensions
const double x0 = 0., y0 = 0., z0 = 0.;
const double dia = 18.0e-4, pitch = 63.0e-4;
const double ampgap = 128.0e-4, driftgap = 2000.0e-4;
const double anodeLZ = 1000.0e-4, cathodeLZ = 1000.0e-4;
double LX = 2*pitch, LY = LX;
double LZ = anodeLZ + ampgap + 2.0*dia + driftgap + cathodeLZ;

// Potentials
const double anodeV = 0., meshV = -500., cathodeV = -700.0;

// Create device
// Note that up down boxes shifted in z to weakly mimic the weaving pattern
SolidBox anode(x0, y0, z0+anodeLZ/2, LX/2, LY/2, anodeLZ/2);
anode.SetBoundaryPotential(anodeV);
anode.SetLabel("anode");
geo.AddSolid(&anode, &Cu);
Code for creating a Micromegas

SolidBox left(x0-pitch/2, y0, z0+anodeLZ+ampgap+dia/2, dia/2, LY/2, dia/2);
left.SetBoundaryPotential(meshV);
geo.AddSolid(&left, &Cu);

SolidBox right(x0+pitch/2, y0, z0+anodeLZ+ampgap+dia/2, dia/2, LY/2, dia/2);
right.SetBoundaryPotential(meshV);
geo.AddSolid(&right, &Cu);

SolidBox up(x0, y0+pitch/2, z0+anodeLZ+ampgap+dia+dia/2, LX/2, dia/2, dia/2);
up.SetBoundaryPotential(meshV);
geo.AddSolid(&up, &Cu);

SolidBox down(x0, y0-pitch/2, z0+anodeLZ+ampgap-dia+dia/2, LX/2, dia/2, dia/2);
down.SetBoundaryPotential(meshV);
geo.AddSolid(&down, &Cu);

SolidBox cathode(x0, y0, z0+anodeLZ+ampgap+2.0*dia+driftgap+cathodeLZ/2, LX/2, LY/2, cathodeLZ/2);
cathode.SetBoundaryPotential(cathodeV);
geo.AddSolid(&cathode, &Cu);
// Parameters for neBEM calculations
double tgtElSize = 10.e-4;
int minEl = 3, maxEl = 7;
int xcopy = 20, ycopy = 20, zcopy = 0;

ComponentNeBem3d nebem;
nebem.SetGeometry(&geo);
nebem.SetNumberOfThreads(2); // Set no. of threads
nebem.SetTargetElementSize(tgtElSize);
nebem.SetMinMaxNumberOfElements(minEl, maxEl);
nebem.SetPeriodicityX(LX);
nebem.SetPeriodicityY(LY);
nebem.SetPeriodicCopies(xcopy, ycopy, zcopy);
nebem.UseLUInversion();
nebem.Initialise();
// Create the sensor.
Sensor* sensor = new Sensor();
sensor->AddComponent(&nebem);
sensor->AddElectrode(&nebem, "anode");
sensor->SetArea(x0-LX, y0-LY, z0+anodeLZ, x0+LX, y0+LY, z0+LZ-cathodeLZ);

// Prepare to draw signal
ViewSignal* signalView = new ViewSignal();
signalView->SetSensor(sensor);
// Create the charge multiplication method
AvalancheMC* avalMC = new AvalancheMC();
avalMC->SetSensor(sensor);
avalMC->EnableSignalCalculation();
avalMC->EnablePlotting(driftView);
// Setup HEED
TrackHeed* track = new TrackHeed();
track->SetSensor(sensor);
track->EnableElectricField();
track->SetParticle("muon"); // Incident particle
track->SetKineticEnergy(1.0e9); // KE of the particle in eV
track->EnablePlotting(driftView);
for(Int_t iEvent=0; iEvent<nEvent;iEvent++) {
    clust_id = 0;
    sensor->ClearSignal(); // Reset signals and free the sensor

    track->NewTrack(track_x, track_y, track_z, tMin, track_dx, track_dy, track_dz);

    bool clust_present=0;
    do {
        // Loop over all the cluster positions along the track
        // number of electrons in present cluster and info of each electron
        int nel;
        double xele, yele, zele, tele, eele, dxele, dyele, dzele;
        clust_present=track->GetCluster(xcls, ycls, zcls, tcls, nel, ecls, extra);
        for(int j = 0; j < nel; j++) { // Loop over all electrons in a cluster
            track->GetElectron(j, xele, yele, zele, tele, eele, dxele, dyele, dzele);
            avalMC->AvalancheElectron(xele, yele, zele, tele);
        }
        clust_id++;
    } while(clust_present!=0);
}}
Build and execute
How does it end?
Lot more data can be generated and analyzed, either offline, or within Garfield++
• Summary and outlook
Summary

• Numerical simulation has been introduced in the context of design and interpretation of gaseous detectors.
• Attempts have been made to get an overview of some of the ways Garfield++ can be used to simulate ionization detectors.
• Various components of the simulation framework have been briefly mentioned.
• Garfield++ installation process has been illustrated.
• Simple gas file example has been built and executed.
• Signal generation of a Micromegas detector has been simulated.
Outlook

- Calculations for gaseous detectors are steadily becoming more detailed. However, much effort is necessary to improve understanding and interpretation.

- Despite the limitations, the tools are mature enough for design purposes. Please note that there are a lot more to explore in and contribute to each of these tools and large number of other options available (next slide).

- Major improvements in physics modelling and computational techniques are necessary to handle more realistic problems.

- Multi-Physics issues can make the work more complex, but rewarding.

- Exciting times ahead!
Major Problems looking for Solutions

• **Volume and surface currents**
  – current through solid-state, dielectric materials, simulation of the VI curve

• **Charging up, Space charge**
  – distortion of field; reduction of ionization probability

• **Discharges**
  – causing permanent damage to detectors, electronics

• **Resistive layers**
  – spark protection, improved spatial resolution

• **Ageing**
  – insulating deposit on anodes and cathodes; formation of strong dipoles, field emissions and micro-discharges – Malter effect
Materials Collected From

- Atsuhiro Ochi
- Blum, Rolandi, Riegler
- Heinrich Schindler
- Igor Smirnov
- Paulo Fonte
- Rob Veenhof
- Steve Biagi
- Werner Riegler

- Many others …
The TEAM

- Deb Sankar Bhattacharya (SINP, IOP, U. Wurzburg)
- Purba Bhattacharya (SINP, NISER, Weizmann, INFN, Cagliari, Calcutta University, Adamas University)
- Sudeb Bhattacharya (SINP, Retd)
- Tanay Dey (VECC)
- Jaydeep Dutta (SINP, Univ. Libre de Bruxelles, Brussles)
- Abhik Jash (SINP, NISER, Weizzmann)
- Raviindrababu Karanam (IITM / TIFR)
- Anil Kumar (IOP / SINP)
- Vishal Kumar
- Nayana Majumdar
- Supratik Mukhopadhyay
- Pralay Das
- Prasant Kumar Rout
- Project students
- Promita Roy
- Subhendu Saha
- Mohammed Salim (AMU / TKM College of Arts & Science)
- Sandip Sarkar
- Muzamil Ahmad Teli (KU)
- Sridhar Tripathy (SINP, York College, City Univ, NY)

Beyond SINP: Rob Veenhof, Heinrich Schindler and members of the RD51 collaboration
Thanks a lot!