Commit 1 : find_all_positions_and_calculate_n2n_delay

只做到計算 end to end delay(Last received time - first transmitted time for the same sequence number)，以及輸出 position(x, y, z)到.txt

Commit 2 : finish_task

完成基本 task

Commit 3 : final

修改 LeoPropagationLossModel 的使用(原本是自己新創一個，改成找到原本在用的 LeoPropagationLossModel)。

Task1: Topology Configuration

1. Set up node positions (latitude, longitude) 分別是 SAT index = 0, 1, 2

```
// LeoLatLong source (20, 0);
// LeoLatLong destination (20, 0);
// LeoLatLong source (6.06692, 73.0213);
// LeoLatLong destination (6.06692, 73.0213);
LeoLatLong source (-16.0634, 142.29);
LeoLatLong destination (-16.0634, 142.29);
```

2. Convert (latitude, longitude) to (x, y, z) coordinates：

In function OutputSatellitePosition_i (順便在裡面算出距離方便後面輸出)

找到地面站和衛星的 node 的 mobility model 並拿到位置輸出

```
118     //地面站的位置
119     Ptr<Node> node = NodeList::GetNode(25);
120     Ptr<MobilityModel> mobility = node->GetObject<MobilityModel>();
121     Vector pos_GA = mobility->GetPosition();
122     // Output (x, y, z) and latitude, longitude
123     outputFile << std::to_string(pos_GA.x) << " "
124            << std::to_string(pos_GA.y) << " "
125            << std::to_string(pos_GA.z) << std::endl;
126
127     //衛星的位置
128     node = NodeList::GetNode(i);
129     mobility = node->GetObject<MobilityModel>();
130     Vector pos_SAT = mobility->GetPosition();
131     // Output (x, y, z) and latitude, longitude
132     outputFile << std::to_string(pos_SAT.x) << " "
133            << std::to_string(pos_SAT.y) << " "
134            << std::to_string(pos_SAT.z) << std::endl;
```

3. Output (x, y, z) to .txt file：

Using std::ofstream

```
std::ofstream outputFile("contrib/leo/examples/satellite_positions_i.txt", std::ios::out);
```

In main:

```
267     initial_position(satellites, 5);
268     double distance = OutputSatellitePosition_i(SATindex);
```

Task2: Calculate Tx Gain

1. Load .txt file to read node coordinates：
2. Use the GS/SAT coordinates to find the elevation angle θ：

```
data = load('position.txt');
% GS 座標
x1 = data(1,1);
y1 = data(1,2);
z1 = data(1,3);
% SAT 座標
x2 = data(2,1);
y2 = data(2,2);
z2 = data(2,3);
```

```
dis_xy = sqrt((x2 - x1)^2 + (y2 - y1)^2);
distance = sqrt((dis_xy)^2 + (z2 - z1)^2);
fprintf("Q1: Euclidean distance: %.6f m\n", distance);

% 計算仰角 θ
theta1_degree = abs( atan2(dis_xy, z2 - z1) * (180 / pi) ); % 轉換成角度
if theta1_degree > 90
    theta1_degree = 180-theta1_degree;
end
```

3. Find the optimal beam θ* based on the elevation angle θ from codebook [0:5:90] (直接找最接近角度的 beam)

```
tx_beam_direction = 0:5:90;
% 找到最接近 theta1 的波束方向
[~, index1] = min(abs(tx_beam_direction - theta1_degree));
op_beam1 = tx_beam_direction(index1);
```

4. Update antenna phase offset • psi = 2 * pi * d * sin (theta)

```
phi_degree = 0.5 : 0.5 : 180;
phi_rad = phi_degree * pi / 180;
psi = 2 * pi * d * sin(phi_rad);
```

(resolutioin 一樣維持一格 0.5 度)

In function steer 也有把 cos 改成 sin

5. Find the Tx gain (跟 lab 做法一樣，without beamforming 的 gain 用 1)

```
a1 = uniform(d, op_beam1, tx_antenna_number);
A1 = dtft(a1, -psi);
gain_table_1 = abs(A1).^2;

rx1_sector_index = round(theta1_degree / 0.5);
% Tx_gain_1 = 1;
Tx_gain_1 = gain_table_1(rx1_sector_index);
```

6. Use Tx gain to calculate pathloss and output it to .txt file

```
pathloss = -friis_equation(freq, Tx_gain_1, 1, distance);
Rx1_power = P_tx_dBm - pathloss;
Rx1 SNR = Rx1 power - N0 dBm;

fileID = fopen('pathloss.txt', 'w'); % 打開文件以寫入
fprintf(fileID, '%f\n', pathloss); % 寫入字符串
fclose(fileID); % 關閉文件
```

Task3: Calculate Rx Power

把 DoCalcRxPower()都從 private 下面移到 public 下面

Replace m_freeSpacePathLoss

```
std::ifstream infile("contrib/leo/examples/pathloss.txt", std::ios::out);
double pathloss;

if (infile.is_open())
{
    infile >> pathloss;
    infile.close();
}
else if (!infile)
{
    std::cerr << "Error opening file!" << std::endl;
}
```

```
// double rxc = txPowerDbm - m_atmosphericLoss - m_freeSpacePathLoss - m_linkMargin;
double rxc = txPowerDbm - m_atmosphericLoss - pathloss - m_linkMargin;
```

Task4: Transmission Configuration

1. Set up transmission configuration

```
double bandwidth_MHz = 2;
double noise_dbm = -110;
double tx_power_dbm = 105.9;
```

2. Call DoCalcRxPower() to get Rx power

使用 DynamicCast 來轉換類型，並從 NetDevice 取得特定的模擬通訊元件，逐步取得 LEO 模組中的信道與路徑損耗模型。

```
Ptr<NetDevice> dev = utNet.Get(0);
Ptr<LeoMockNetDevice> mockDev = DynamicCast<LeoMockNetDevice>(dev);
Ptr<LeoMockChannel> channel = DynamicCast<LeoMockChannel>(mockDev->GetChannel());
Ptr<LeoPropagationLossModel> lossModel = DynamicCast<LeoPropagationLossModel>(channel->GetPropagationLoss());
Ptr<MobilityModel> txMobility = users.Get(0)->GetObject<MobilityModel>();
Ptr<MobilityModel> rxMobility = satellites.Get(SATindex)->GetObject<MobilityModel>();

double rx_power_dbm = lossModel->DoCalcRxPower(tx_power_dbm, txMobility, rxMobility);
```

3. Calculate SNRdB　並轉呈 ratio

```
double rx_power_dbm = lossModel->DoCalcRxPower(tx_power_dbm, txMobility, rxMobility);
double SNR_dB = rx_power_dbm - (noise_dbm);
double SNR_ratio = pow(10, SNR_dB / 10);
```

4. Calculate data rate based on Shannon capacity

```
double Shannon_Capacity_Mbps = bandwidth_MHz * log2(1 + SNR_ratio);
```

Task5: Compute E2E Delay

1. Change link data rate settings

   轉成字串的形式再放進去

```
    ////////////////////////////////////////////////////////////////////////
    std::ostringstream oss;
    oss << Shannon_Capacity_Mbps << "Mbps";
    utNet.Get(25)->GetObject<MockNetDevice>()->SetDataRate(DataRate(oss.str()));
    utNet.Get(SATindex)->GetObject<MockNetDevice>()->SetDataRate(DataRate("1Gbps"));
    ////////////////////////////////////////////////////////////////////////
```

2. Output end-to-end delay

```
uint32_t seq = header.GetSequenceNumber().GetValue();
Time now = Simulator::Now();

// 解析發送時間與到達時間
if (context.find("Tx") != std::string::npos) {
    if (sendTimes.find(seq) == sendTimes.end()) { // 第一次出現這個序列號
        sendTimes[seq] = now;
    }
} else if (context.find("Rx") != std::string::npos) {
    arrivalTimes[seq] = now;

    // 計算 end-to-end 延遲
    if (sendTimes.find(seq) != sendTimes.end()) {  //sendTimes有紀錄了
        Time delayTime = arrivalTimes[seq] - sendTimes[seq];
        delay[seq] = delayTime.GetSeconds();
        cout << "seq: " << seq << "   n2n delay: " << delay[seq] << std::endl;
    }
}
```

Sendtime 裡面沒有該 seq number 的紀錄時，再記錄 sendtime(第一次 Tx)是 Rx 的話，就更新 end to end delay，直到結束就會是用最後的 Rx 來算時間