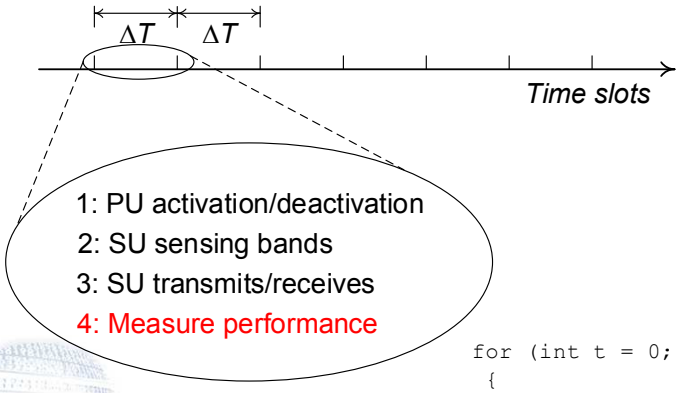
Project I: Simulation Framework for CRNs v2.0

Dr. Mohammed Hawa
Electrical Engineering Department
The University of Jordan

Project I

- Build a simulation framework to test cognitive radio networks.
- The framework allows SUs to perform sensing and spectrum allocation, then measures the resulting performance parameters.
- Use C/C++ and MS Visual Studio 2013.
- Include the probabilistic method implementation and results (see paper #1 and paper #2).
- LATER: Your proposed algorithm: can be sensing, spectrum allocation, or combination of both.
- Output is a group of plots for the various performance parameters.

Assume Slotted System




```
for (int t = 0; t < SLOTS; t++)
 {
 .
 .
 .
 }
```


Simulation Parameters

- Number of secondary users (SUs) $N = 10$
- Number of spectrum bands $M = 100$
- One primary user per band.
- So, number of primary users (PUs) $= M$
- Number of time slots $T = 20{,}000$
- Demand $\hat{S}_n = 5 - 15$
- So, Load $L = 50\text{–}150\%$
- Later we can vary the load or change the parameters.

Spectrum Sensing

- PU is activated/deactivated in its band randomly with probability 0.1/0.9, or in a deterministic was at T_{start}.
- Sensing is not perfect. Assume:
- Probability of **false alarm:**

$$P_{FA} = \Pr[\Gamma > \gamma \mid \mathcal{H}_0] = 0.1$$

- Probability of **miss-detection:**

$$P_{MD} = \Pr[\Gamma \leq \gamma \mid \mathcal{H}_1] = 0.1$$

Flipping Coins

```
rand()
// integer value between [0, RAND_MAX]

double RandomNumber =
       double(rand()) / double(RAND_MAX);
       // uniform value between [0.0, 1.0]

if (RandomNumber <= 0.7)
  do something; // probability 0.7
else
  do something else; // probability 0.3
```

Spectrum Allocation

- For the probabilistic technique:
- Each SU selects $\lceil \hat{S}_n \rceil$ randomly (with equal probability) of the M available spectrum bands for the current time slot.
- No dependence on sensing (no intelligence).
- Sensing here is detecting collisions that happen after transmitting data.
- There is no coordination with other SUs.
- SU[n] can only hope it was lucky that no one else chose the same bands it has selected (causing a collision).

Suggestion

- Use `std::vector`.
- Easier memory management than arrays or two-dimensional arrays. For example:
- `vector <vector <int> > SU;`
- `vector <list <int> > SU;`
- Each SU records it allocated bands:
- `SU[0] = {1, 25, 5, 88, 17}`
- `SU[3] = {40, 5, 65, 14, 79}`

Performance Parameters

- BS successful transmission: $s_n(t)$
- BS contention bands: $c_n(t)$
- PU interference time: T_i
- Settling time: T_s
- Utilization: U
- Throughput: Y
- I will provide the code for plotting a graph from x-y data (similar to MATLAB `plot`).

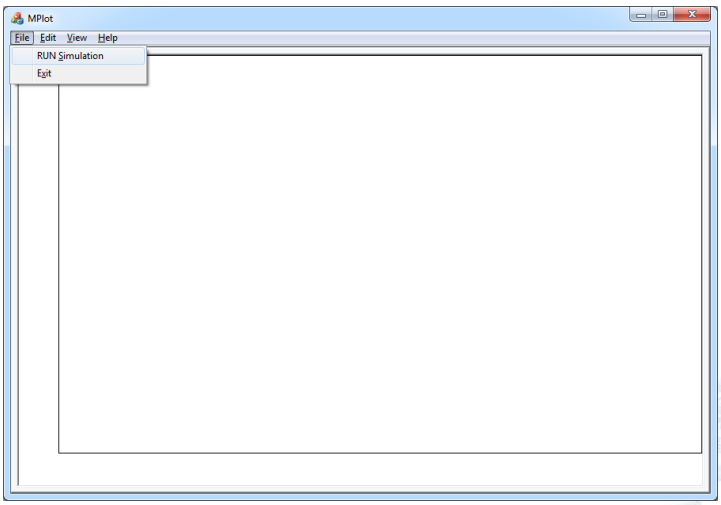
Submission

- Each team will submit the code as a team.
- I will test the code and then ask the team members about the code decisions.
- Team members should meet regularly, and distribute the load amongst themselves.
- No inter-Team cooperation WHATSOEVER!
- Similar solutions with receive zero credit.

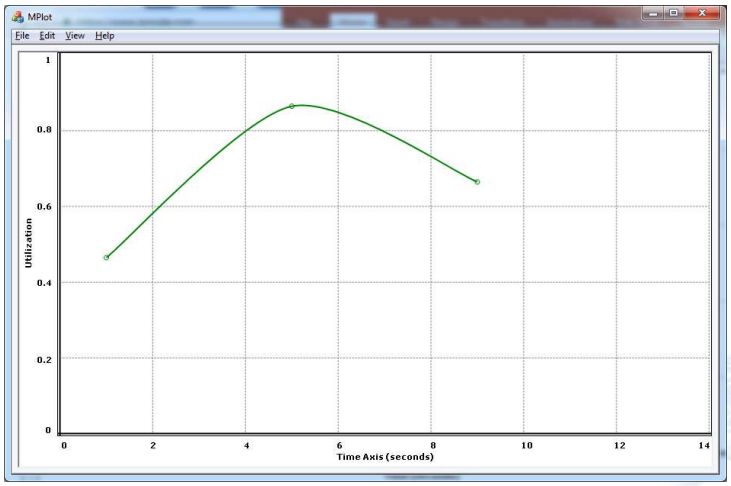
MPlot: Plotting Framework

- Extension of an open source project.
- I modified the container, and added x- and y-axes.
- If you detect a bug, please report it back to me, or you can fix the code yourself.
- Will try to introduce MATLAB-similar syntax later.

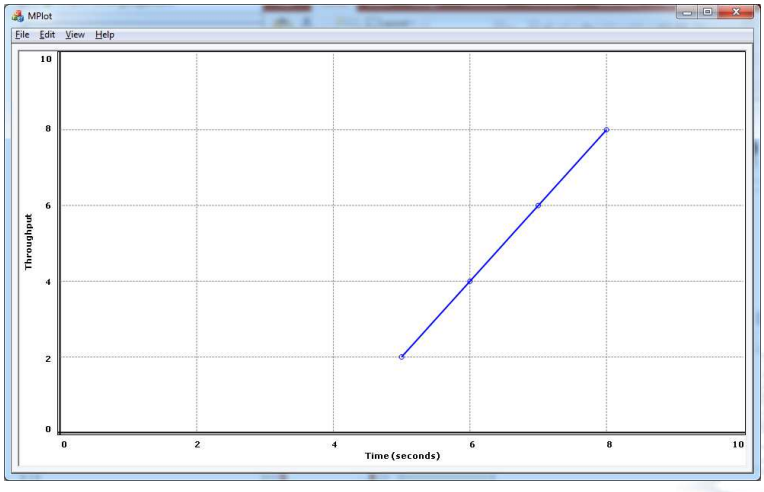
Press RUN Simulation



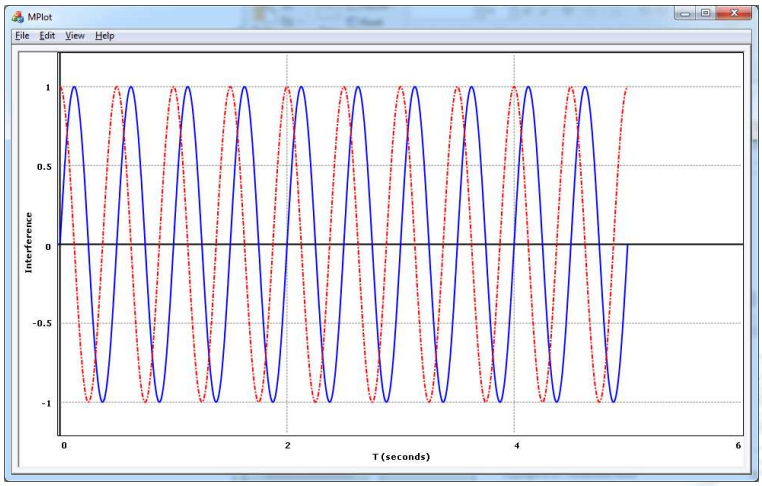
Your Code Should Work



Examples Are Shown



See AddPlot Syntax



MPlot.cpp

```
void MPlotApp::OnRunSimulation()
{
	// TODO: Run your simulation code from HERE

	// record the RESULTS then you can call as many plots as you need
	MainFrame* frame1 = new MainFrame;
	frame1->LoadFrame(IDR_MAINFRAME,
			WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, NULL, NULL);

	...

	MainFrame* frame3 = new MainFrame;
	frame3->LoadFrame(IDR_MAINFRAME,
			WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, NULL, NULL);

	frame3->m_figure->m_axes.xLabel(_T("T (seconds)"));
	frame3->m_figure->m_axes.yLabel(_T("Interference"));
	frame3->m_figure->m_axes.xLimit(0, 6.0);
	frame3->m_figure->m_axes.yLimit(-1.2, 1.2);

	// Calculate period: four for nPntNmb
	double minX = 0.0f, deltaX = 0.01f, maxX = 5.0f;
	int points = (int)ceil((maxX - minX) / deltaX);

	V_CHARTDATAD vSineData, vCosineData;
	vSineData.resize(points, PointD(0.0, 0.0));
	vCosineData.resize(points, PointD(0.0, 0.0));
```



```
double frequency = 2 * 3.14159265358979 * 2;
// Fill the vector
int i = 0;
for (double X = minX; X <= maxX; X += deltaX)
{
	double Y = sin(frequency * X);
	vSineData[i] = PointD(X, Y);
	vCosineData[i] = PointD(X, cos(frequency * X));
	i++;
}

frame3->m_figure->AddPlot(true, true, string_t(_T("Sin")), 5,
	DashStyleSolid, 2.0f, 0.4f,
	Color(ARGB(Color::Blue)), vSineData, true);
frame3->m_figure->AddPlot(true, true, string_t(_T("Cosine")), 5,
	DashStyleDashDot, 2.0f, 0.4f,
	Color(ARGB(Color::Red)), vCosineData, true);

frame3->m_figure->ShowDataLegend(5.0);
frame3->m_figure->ShowNamesLegend();
//m_Figure->ShowDataView(GetChartName(2), true);
frame3->m_figure->ShowAxisXBoundaries(true, true);
frame3->m_figure->EnableUser(true);
frame3->ShowWindow(SW_SHOW);
frame3->UpdateWindow();
m_aryFrames.Add(frame3->GetSafeHwnd());
```


PU interference time: T_i

- Requires consistent interference on PU.
- Your algorithm might not have that.
- Hence, replace PU interference time T_i with PU interference ratio (per PU) ψ_m.
- Say you have a total of 20,000 time slots, during which PU_7 is active for 8,000 slots. If PU_7 suffers interference during 100 slots, then $\psi_7 = \frac{100}{8000} = 0.0125 = 0.125\%$

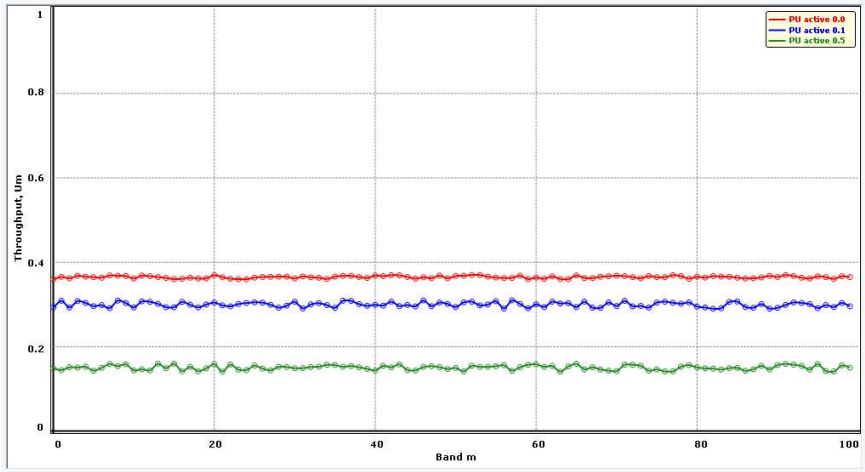
Project 1 Submission

- Submit your source code (including project, cpp, h, … files) on a CD or flash drive.
- One project per team: on Thursday March 30, 2017.
- The code must compile on VS2013 without errors.
- The program should perform: PU activation, unreliable sensing, SU allocation, and performance measurement.
- Running the program should result in the following 7 figures.

Throughput

- Throughput Y_m for each band averaged over all time slots Versus band number.
- Three curves: For different PU activation probability = 0.0, 0.1, 0.5
- Use different colors and use a Legend.
- $Y_m^r = 1$ for band m in time slot r if band is successfully utilized by only one SU, and 0 if band is empty, used by many SUs or used by both SU(s) and PU.
- $Y_m = \frac{\sum_{r=1}^{R} Y_m^r}{R}$

Not necessarily actual results…



Utilization

- Utilization U_m for each band averaged over all time slots Versus band number.
- Three curves: For different PU activation probability = 0.0, 0.1, 0.5
- Use different colors and use a Legend.
- $U_m^r = 1$ for band m in time slot r if band is used by one (or more) SUs, or 0 if band is empty or has only a PU.
- $U_m = \frac{\sum_{r=1}^{R} U_m^r}{R}$

SU Successful Transmission

- Successfully acquired bands S_n for each SU averaged over all time slots Versus SU number.
- Three curves: For different PU activation probability = 0.0, 0.1, 0.5
- Use different colors and use a Legend.
- s_n^r = number of bands successfully acquired by only SU n during time slot r without collisions from other SUs or PUs.
- $S_n = \frac{\sum_{r=1}^{R} s_n^r}{R}$

SU Collisions

- Collision bands C_n for each SU averaged over all time slots Versus SU number.
- Three curves: For different PU activation probability = 0.0, 0.1, 0.5
- Use different colors and use a Legend.
- c_n^r = number of bands acquired by SU n during time slot r that suffer collision from SUs or PUs.
- $C_n = \frac{\sum_{r=1}^{R} c_n^r}{R}$

PU Interference Ratio

- Interference Ratio ψ_m for each PU (band) Versus PU (band) number.
- Three curves: For different PU activation probability = 0.0, 0.1, 0.5
- Use different colors and use a Legend.
- Say you have a total of 20,000 time slots, during which PU m is active for 8,000 slots. If PU m suffers interference (from one or more SUs) during 100 slots, then ψ_m= 100/8000 = 0.0125

Successful versus Time

- Successfully acquired bands s_7^r for SU 7 Versus time slots.
- One curve: Deterministic PU activation. Exactly in the middle of the simulation PU 0 to 49 become active. PU 50 to 99 stay inactive for the whole simulation.
- s_n^r = number of bands successfully acquired by only SU n during time slot r without collisions from other SUs or PUs.

Changing Load

- First 6 figures should be done using $\hat{S}_n = 10$
- Finally draw: Successfully acquired bands S (averaged over all time slots and all SUs) Versus $\hat{S}_n$.
- Three curves: For different PU activation probability = 0.0, 0.1, 0.5
- Use different colors and use a Legend.
- $S = \frac{\sum_{r=1}^{R} \sum_{n=1}^{N} s_n^r}{R \times N}$

