

Ćwiczenie - tryger RPC

Robert Boniecki

5 września 2012

Streszczenie

W ramach ćwiczenia utworzono procedurę do analizy danych wykorzystując oprogramowanie wykorzystywane w eksperymencie CMS - CMSSW. Potwierdzono poprawność procedur numerujących warstwy systemu RPC, oraz stworzono zbiór wykresów zależności różnicy kąta ϕ względem pędu poprzecznego mionu.

1 Cel ćwiczenia

Ćwiczenie polegało na stworzeniu procedury do analizy danych przy użyciu środowiska CMSSW. Celem procedury była analiza danych pochodzących z symulacji przypadków z pojedynczym mionem. W pierwszej części ćwiczenia sprawdzona została poprawność procedury sprawdzającej dopasowanie punktów w płaszczyźnie (z, R) w których mion zostawiał sygnał, do odpowiednich warstw systemu RPC detektora CMS. Następnie wykonane zostały wykresy zależności różnic kąta ϕ pomiędzy sygnałami w kolejnych warstwach, względem pędu poprzecznego mionu. Różnica $\Delta\phi$ liczona była na trzy różne sposoby:

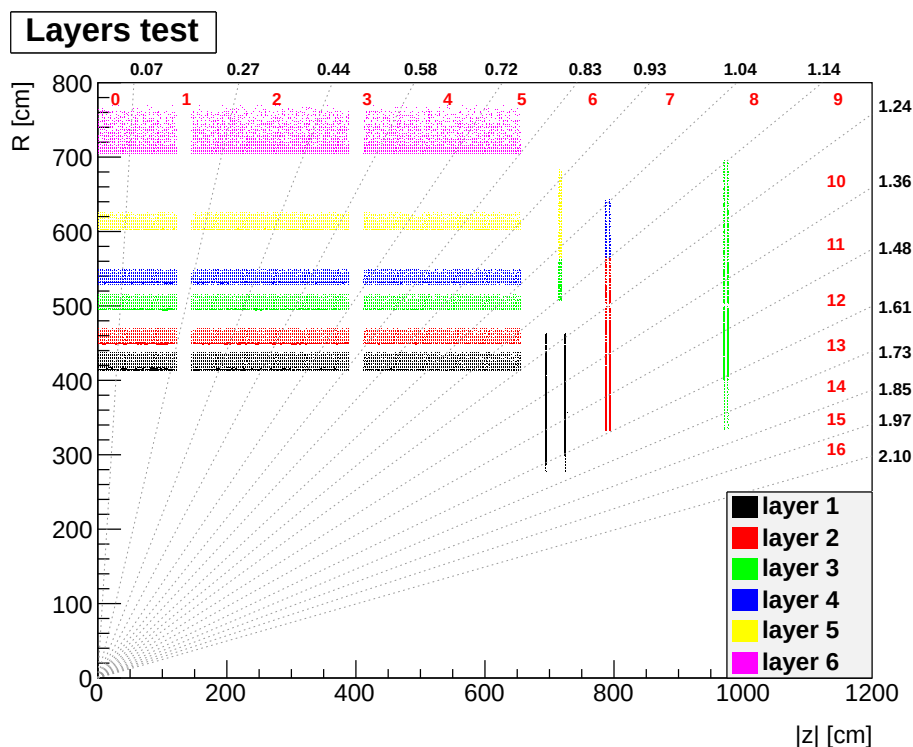
- przedstawiając kąt ϕ w postaci zmiennoprzecinkowej,
- przyporządkowując kątowi ϕ numer paska w danej warstwie RPC,
- symulując odpalenie klastra pasków przy przejściu mionu przez daną warstwę, i obliczając minimalne $\Delta\phi$ dla klastrów.

2 System RPC

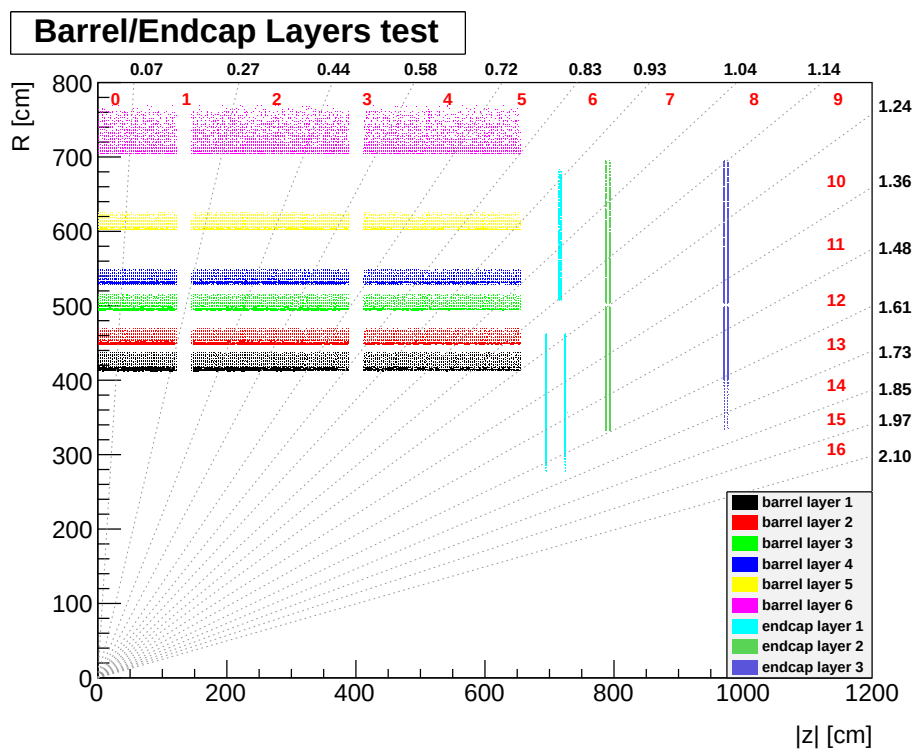
System detektorów RPC (Resistive Plate Chambers) detektora CMS składa się z sześciu warstw w obszarze beczki i z trzech warstw w obszarach pokryw. Warstwy komór RPC są podzielone na paski, po 1152 paski w pełnym kącie azymutalnym. System RPC jest podstawą układu wyzwalania, pozwala na bardzo szybkie oszacowanie pędu poprzecznego mionu poprzez porównanie wzoru utworzonego przez kolejno odpalone segmenty z kombinacjami wzorcowymi, odpowiadającymi kolejnym przedziałom pędu. System RPC sięga do $|\eta| < 2.1$, cały przedział η podzielony jest na 16 wież.

3 Sprawdzenie procedury numerującej warstwy

W ramach ćwiczenia sprawdzona została poprawność procedury numerującej warstwy zarówno z punktu widzenia nadlatującego mionu, jak i osobno dla obszaru beczki i pokryw. Procedura jako argument bierze wartość pseudopospieszności η odpowiadającej segmentowi w którym mion dał sygnał, a zwraca numer warstwy. Wykresy sprawdzające poprawność tych procedur przedstawiono na rysunkach 1 i 2.

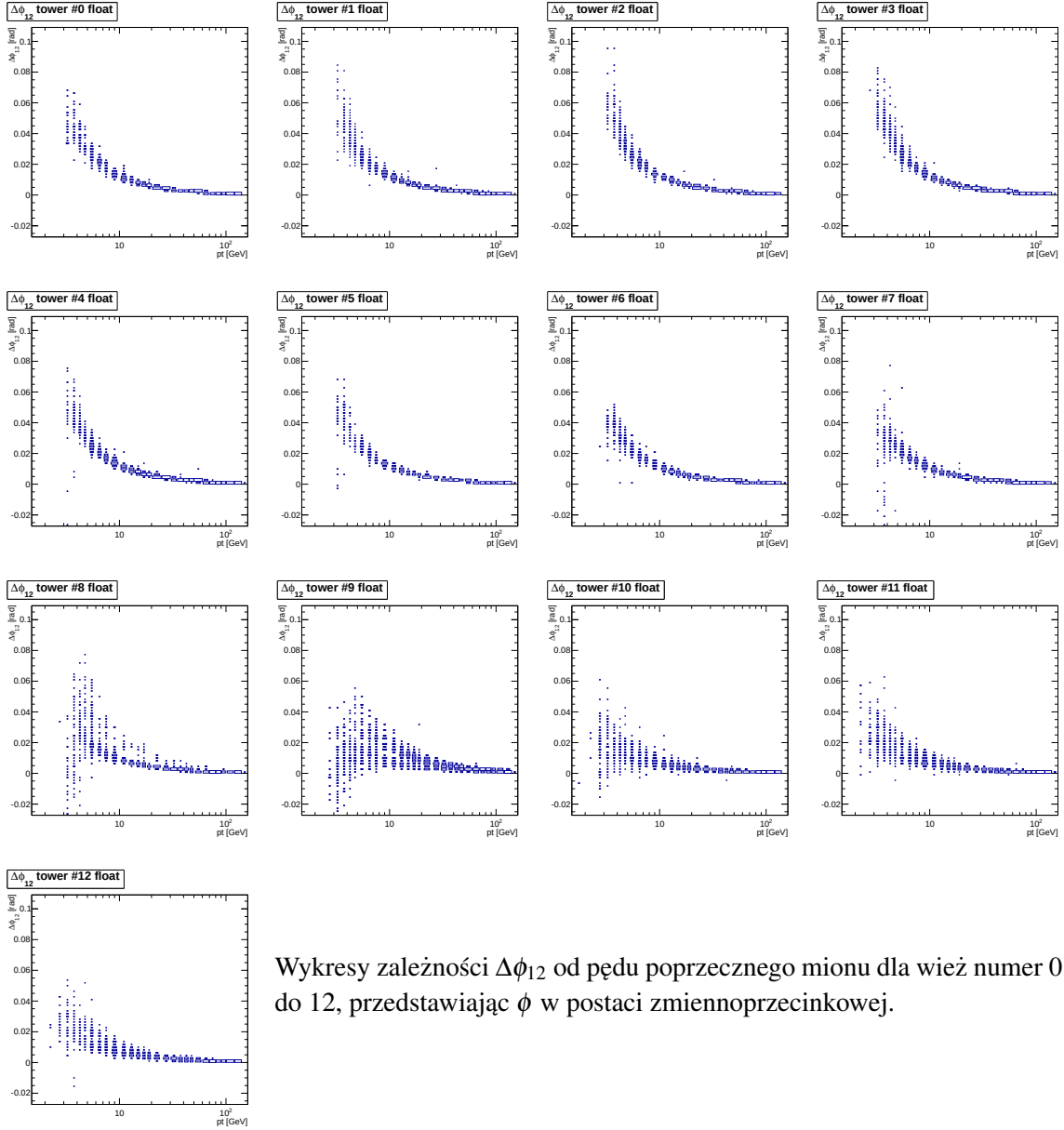


Rysunek 1: Wykres przedstawiający punkty w płaszczyźnie (z, R) w których zarejestrowano sygnały od mionów wraz z odpowiadającymi im kolejnymi warstwami z punktu widzenia nadlatującego mionu.

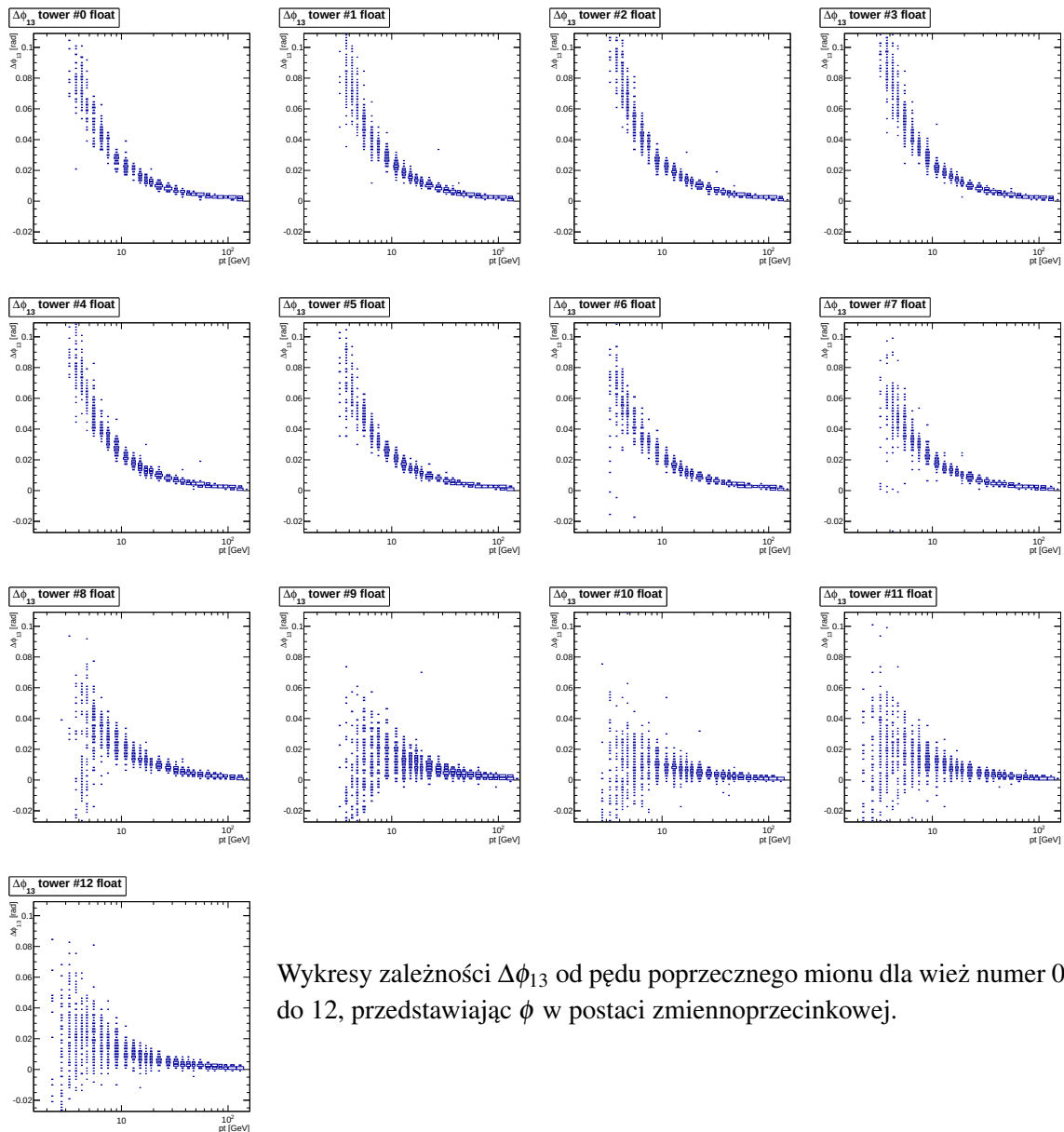


Rysunek 2: Wykres przedstawiający punkty w płaszczyźnie (z, R) w których zarejestrowano sygnały od mionów wraz z odpowiadającymi im warstwami osobno w obszarze beczki osobno w obszarze pokryw..

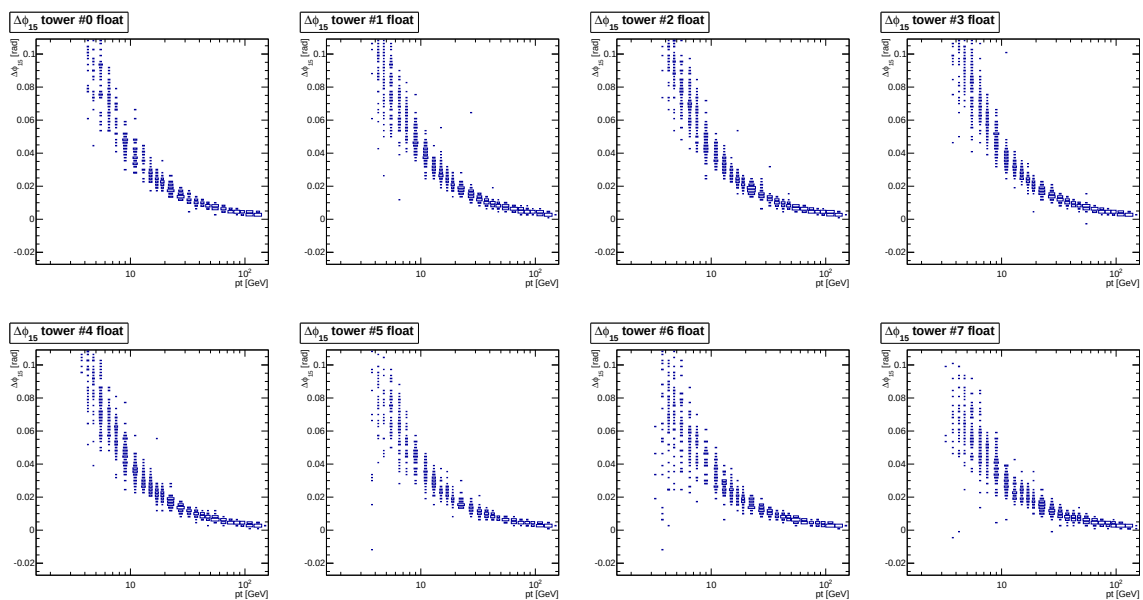
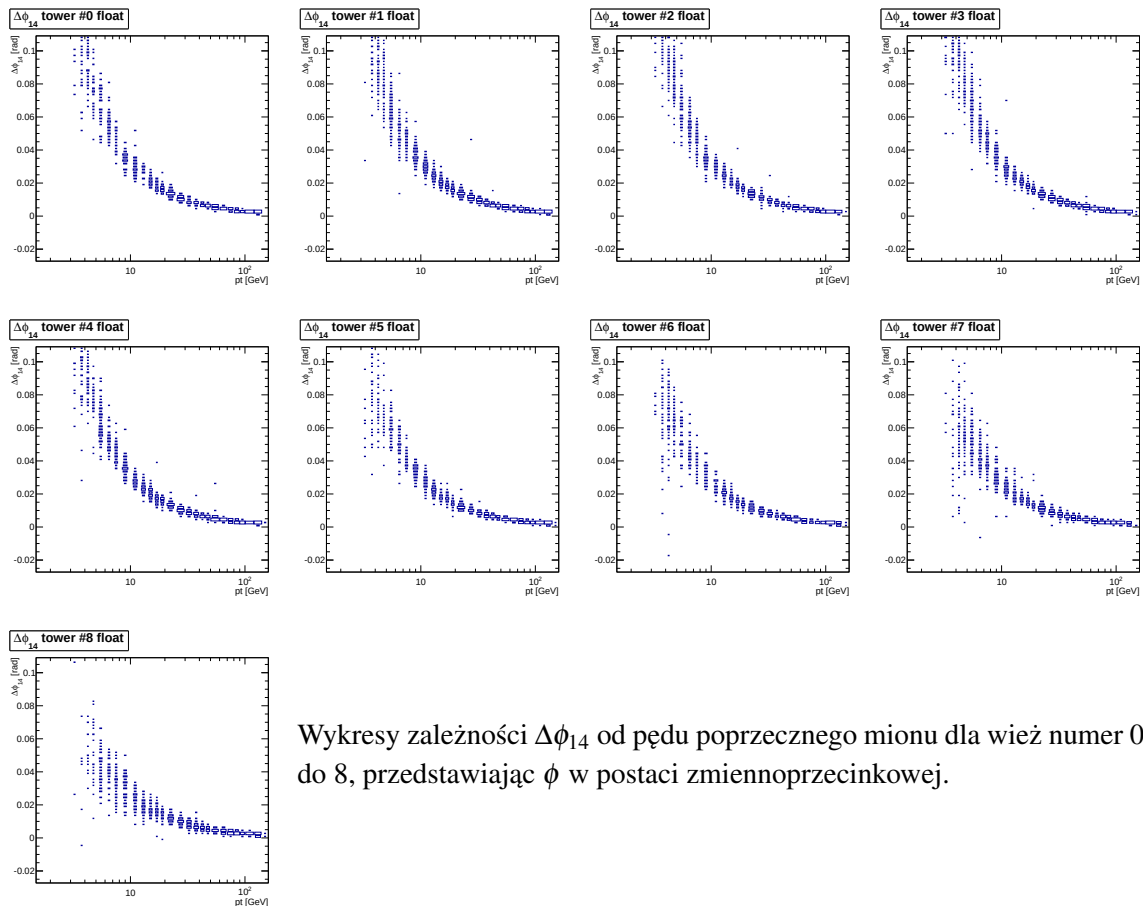
4 Wykresy zależności $\Delta\phi$ od pędu poprzecznego mionu.

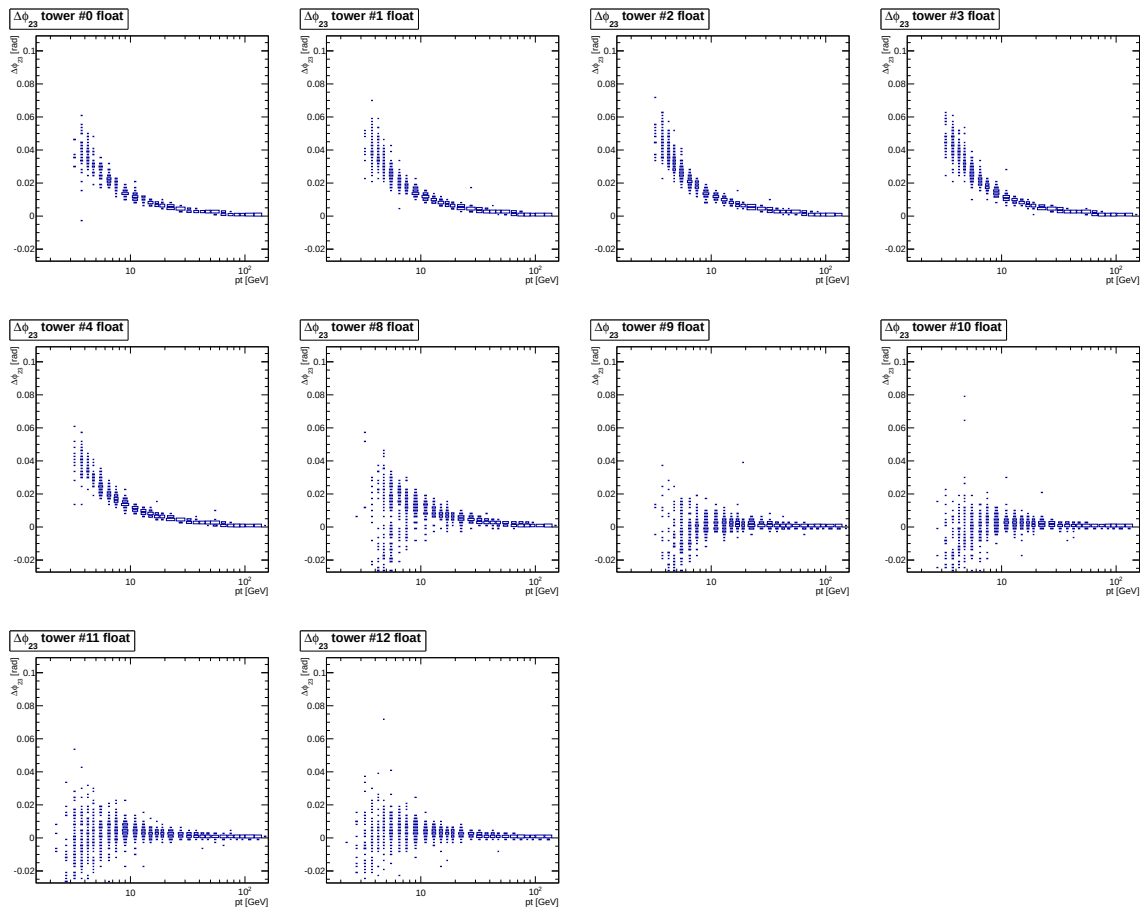


Wykresy zależności $\Delta\phi_{12}$ od pędu poprzecznego mionu dla wież numer 0 do 12, przedstawiając ϕ w postaci zmiennoprzecinkowej.

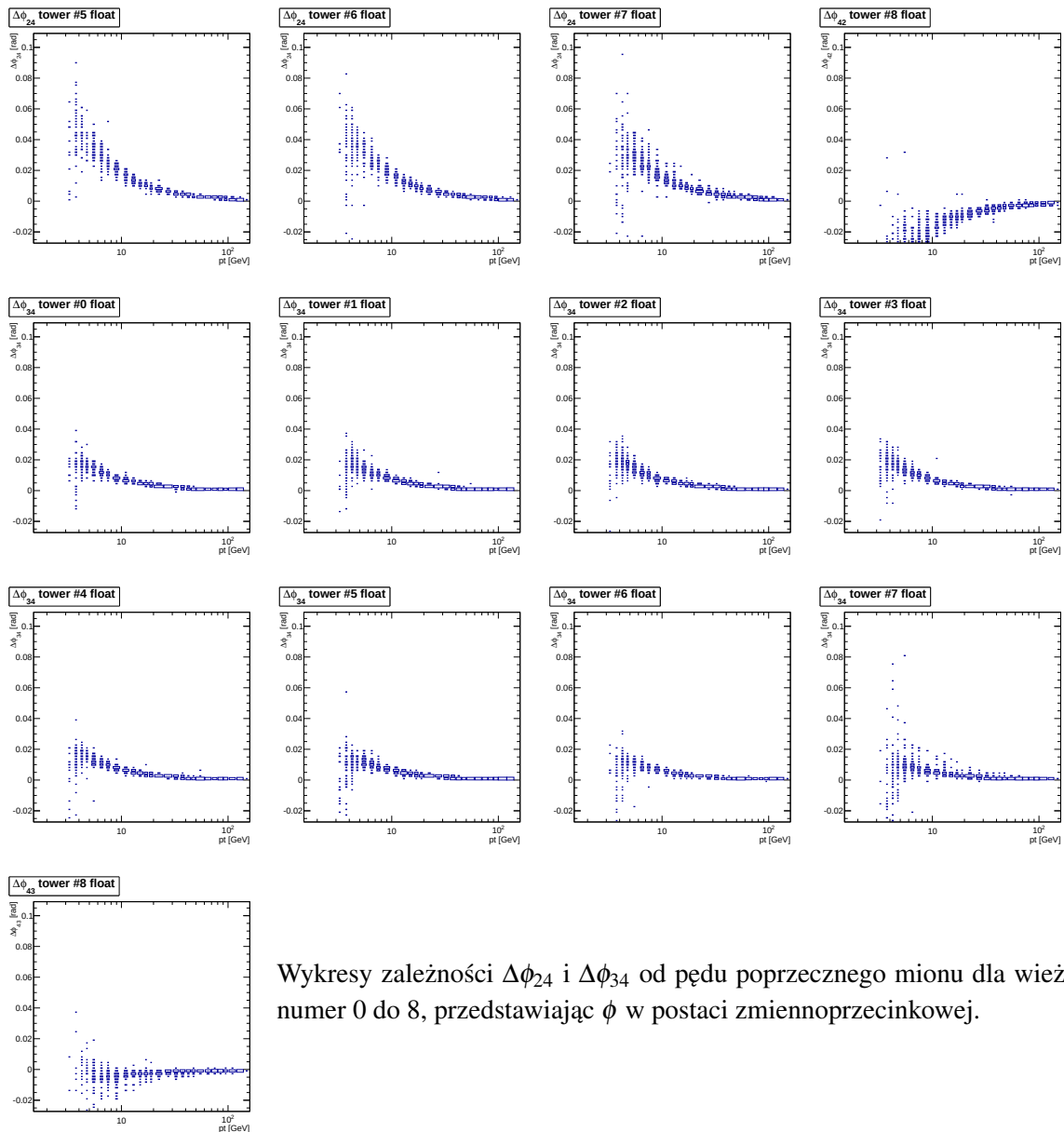


Wykresy zależności $\Delta\phi_{13}$ od pędu poprzecznego mionu dla wież numer 0 do 12, przedstawiając ϕ w postaci zmiennoprecinkowej.

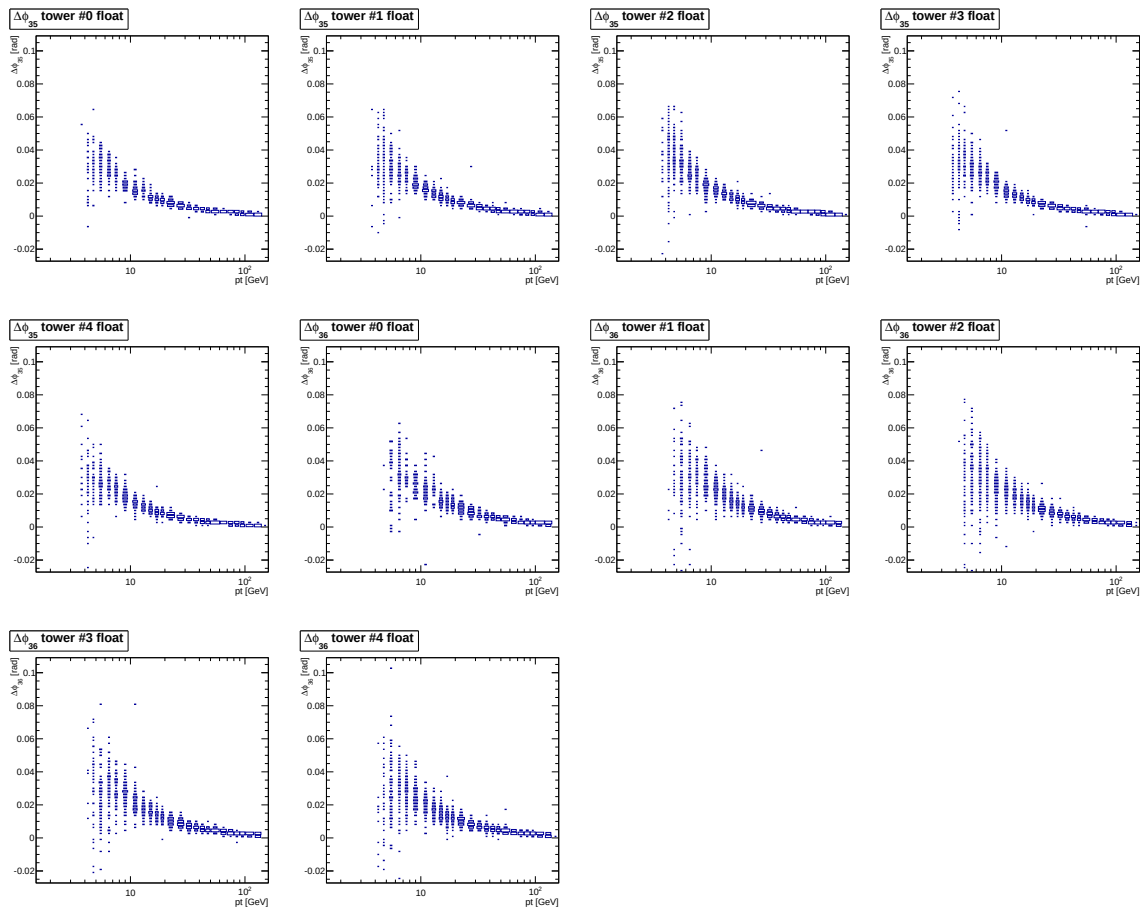




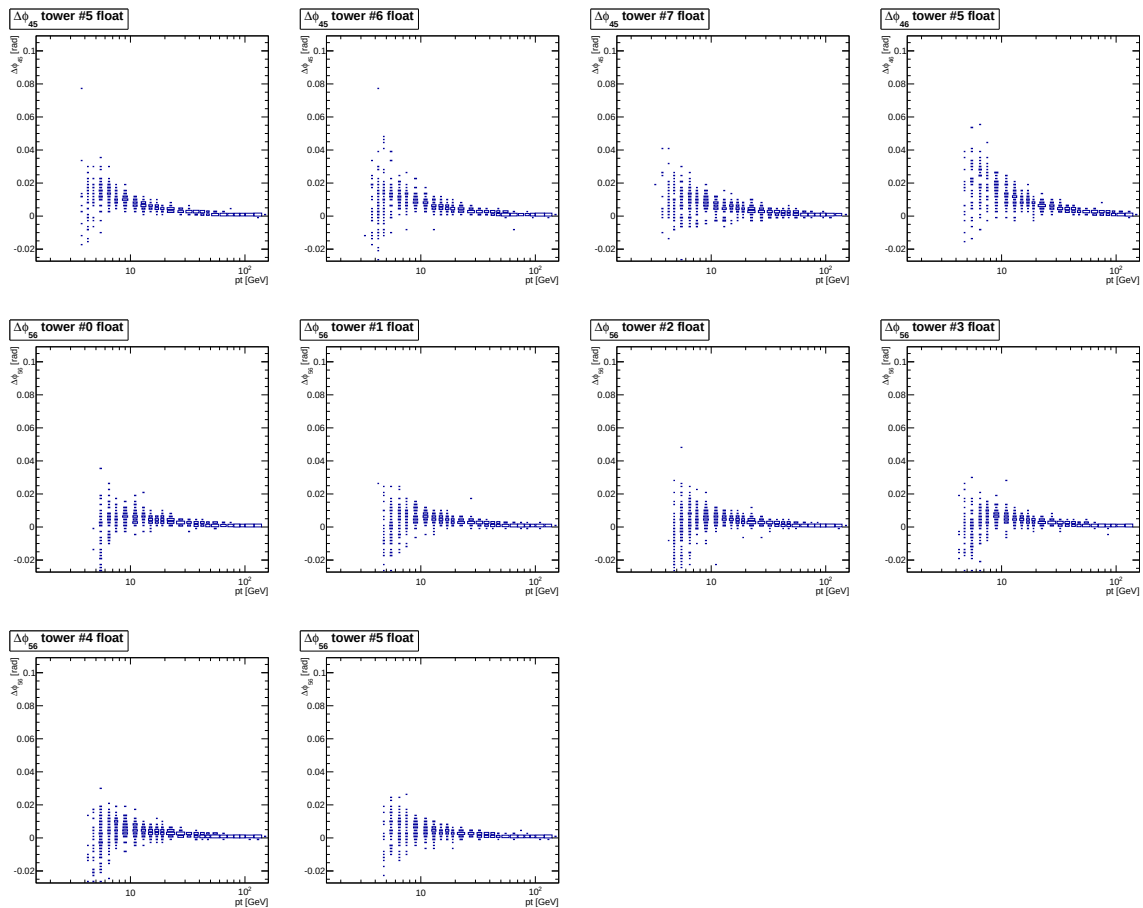
Wykresy zależności $\Delta\phi_{23}$ od pędu poprzecznego mionu dla wież numer 0 do 12, przedstawiając ϕ w postaci zmiennoprzecinkowej.



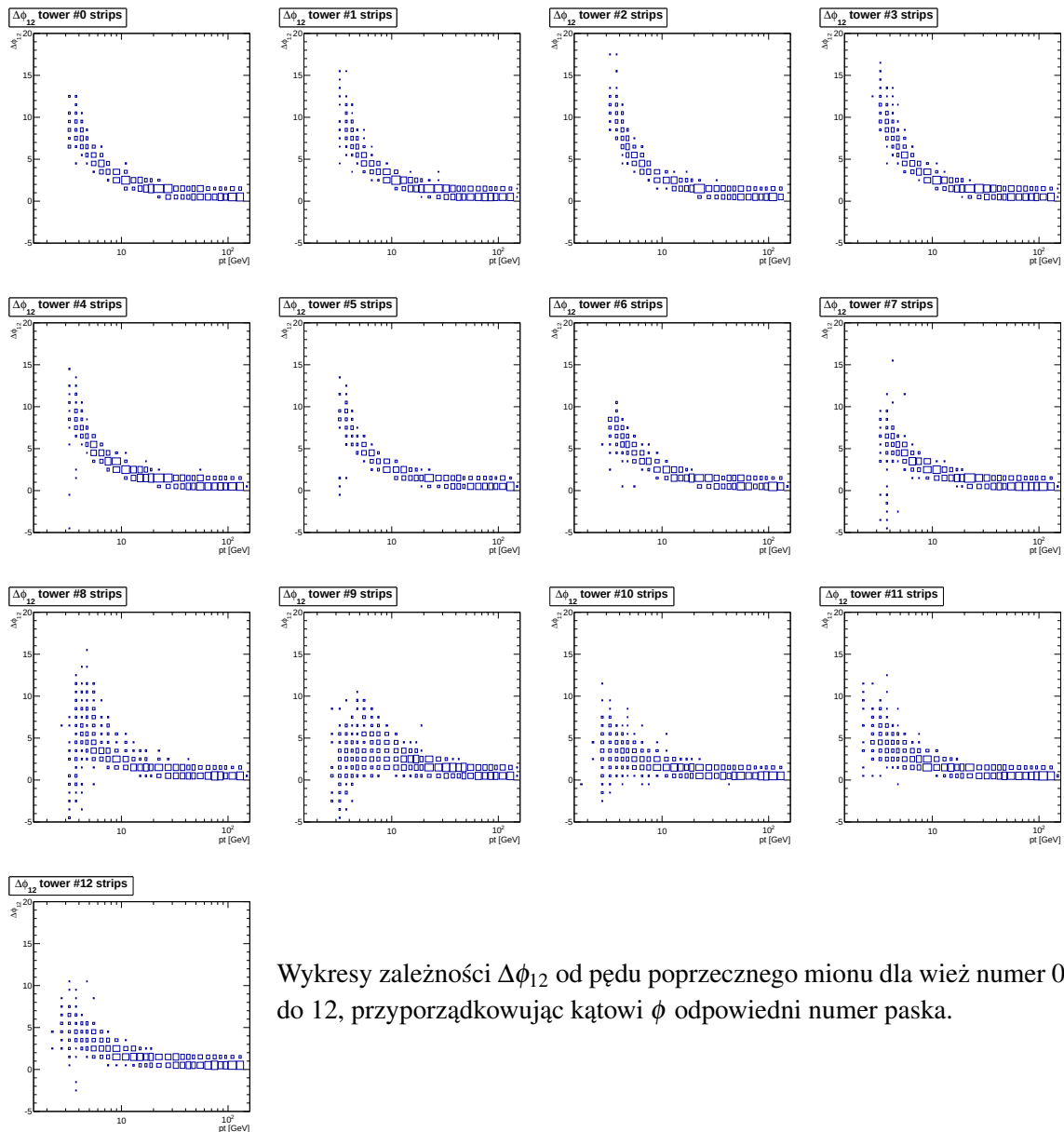
Wykresy zależności $\Delta\phi_{24}$ i $\Delta\phi_{34}$ od pędu poprzecznego mionu dla wież numer 0 do 8, przedstawiając ϕ w postaci zmiennoprzecinkowej.



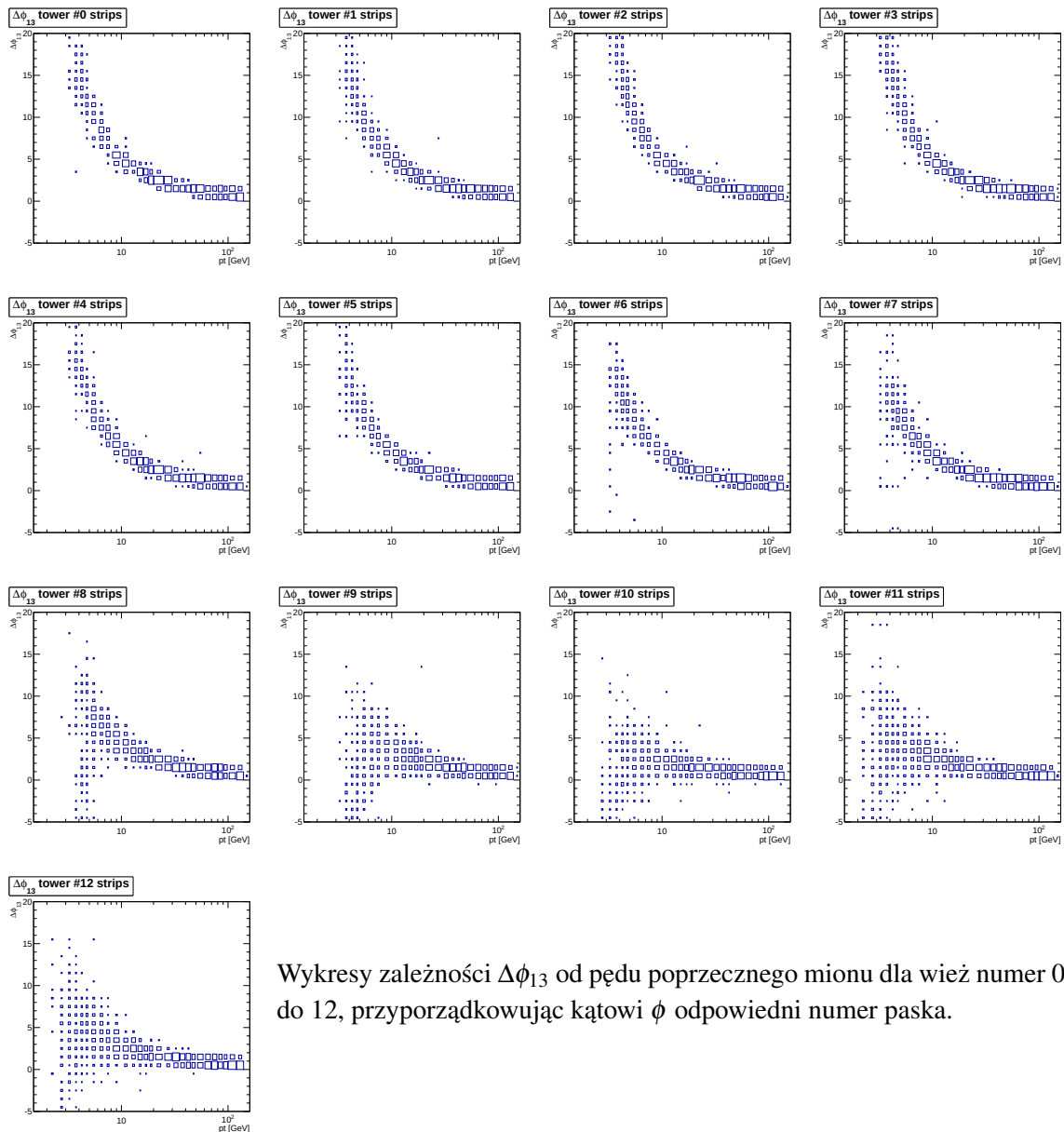
Wykresy zależności $\Delta\phi_{35}$ i $\Delta\phi_{36}$ od pędu poprzecznego mionu dla wież numer 0 do 4, przedstawiając ϕ w postaci zmiennoprzecinkowej.



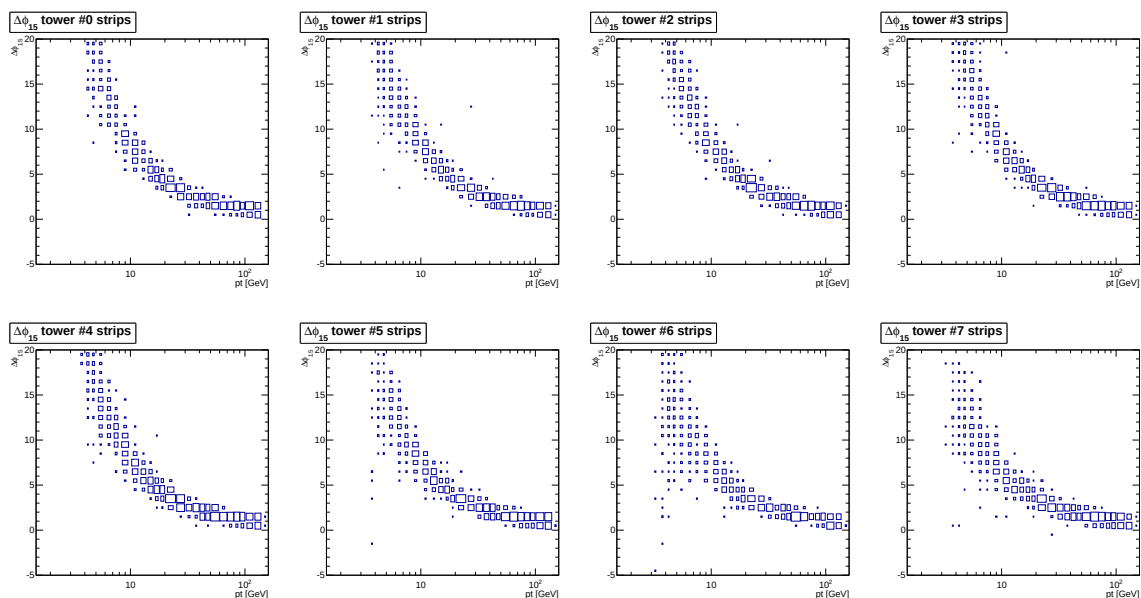
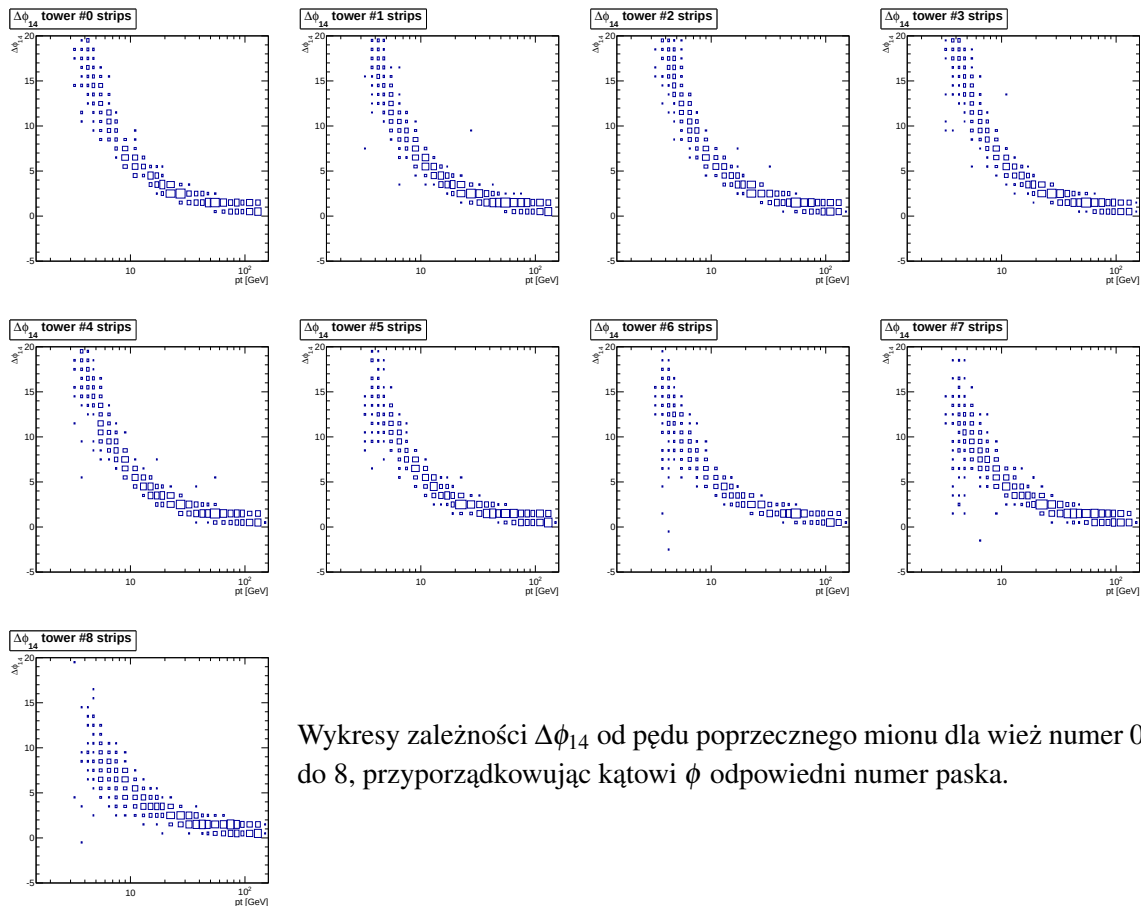
Wykresy zależności $\Delta\phi_{45}$, $\Delta\phi_{46}$ i $\Delta\phi_{56}$ od pędu poprzecznego mionu dla wież numer 0 do 7, przedstawiając ϕ w postaci zmiennoprzecinkowej.

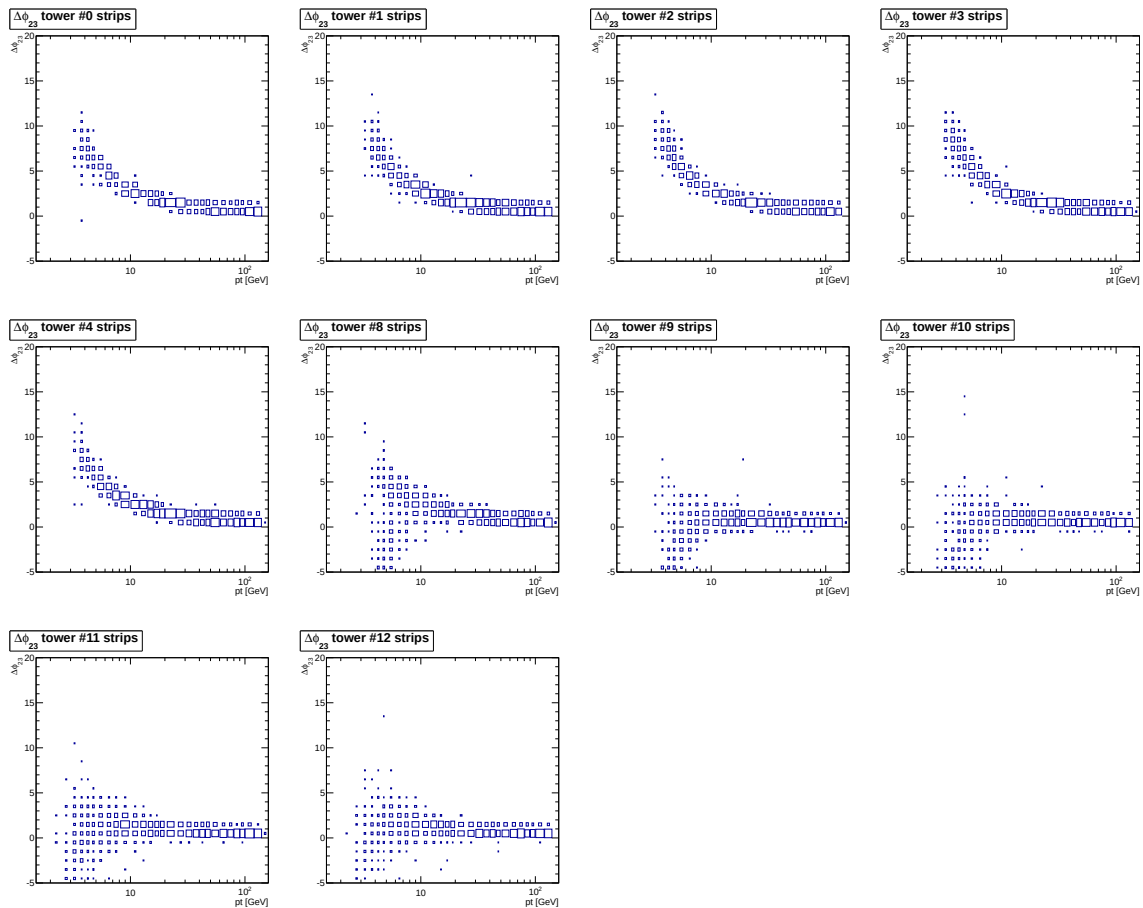


Wykresy zależności $\Delta\phi_{12}$ od pędu poprzecznego mionu dla wież numer 0 do 12, przyporządkowując kątowi ϕ odpowiedni numer paska.

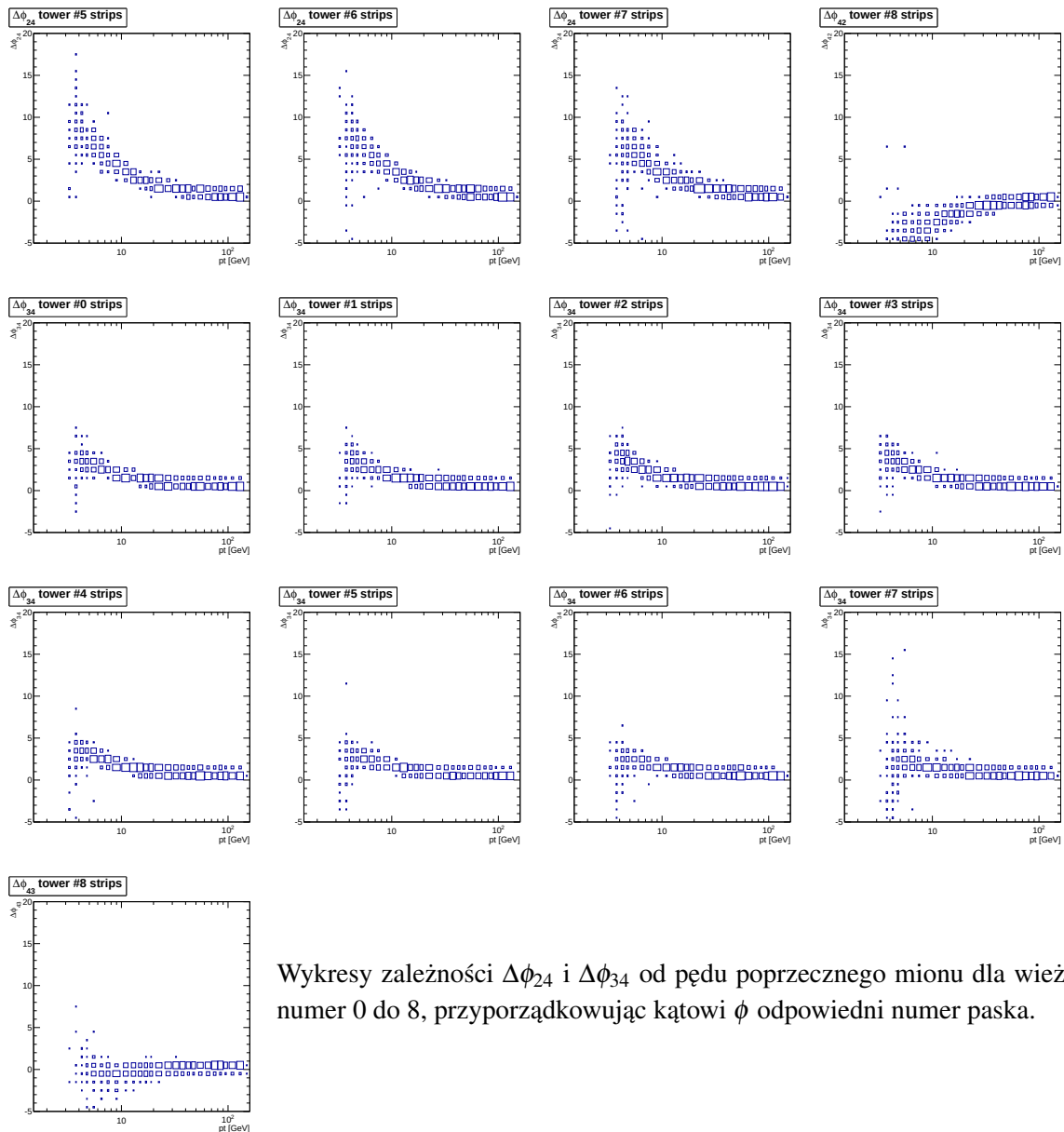


Wykresy zależności $\Delta\phi_{13}$ od pędu poprzecznego mionu dla wież numer 0 do 12, przyporządkowując kątowi ϕ odpowiedni numer paska.

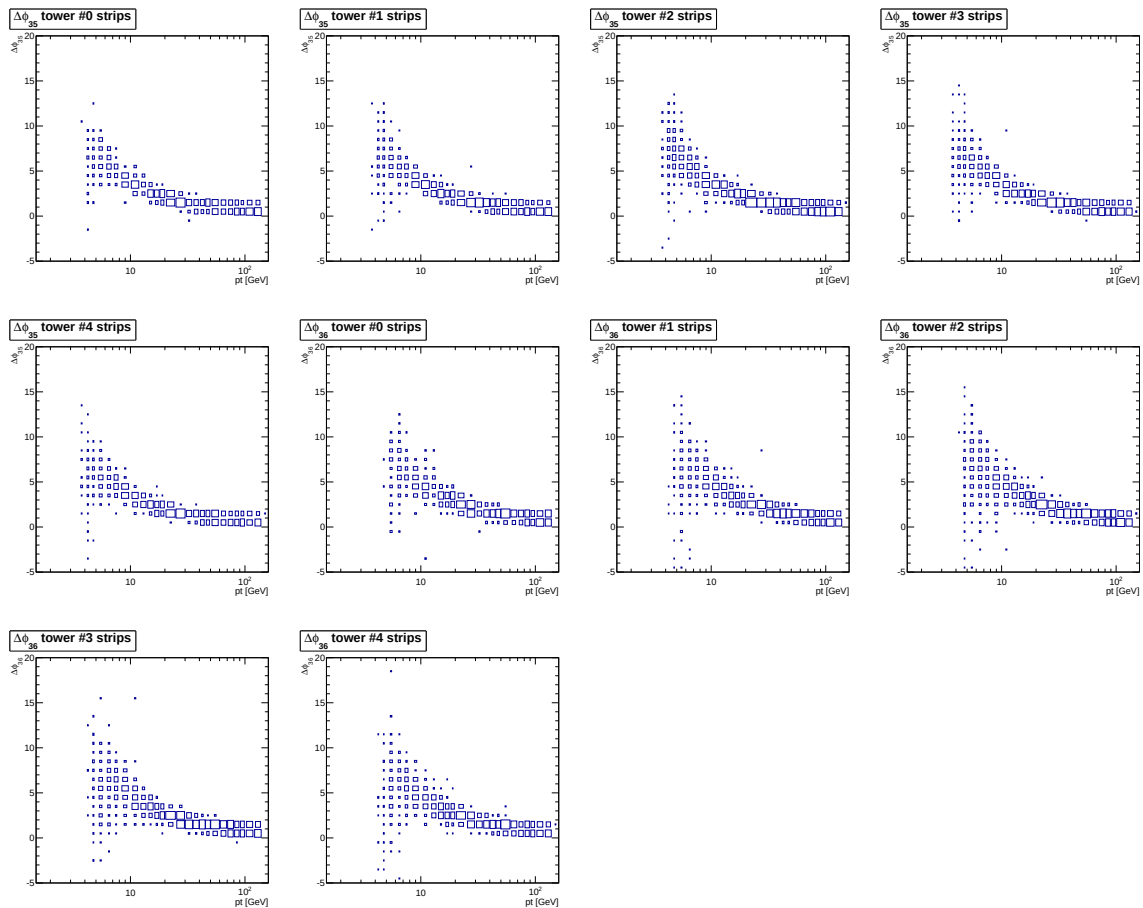




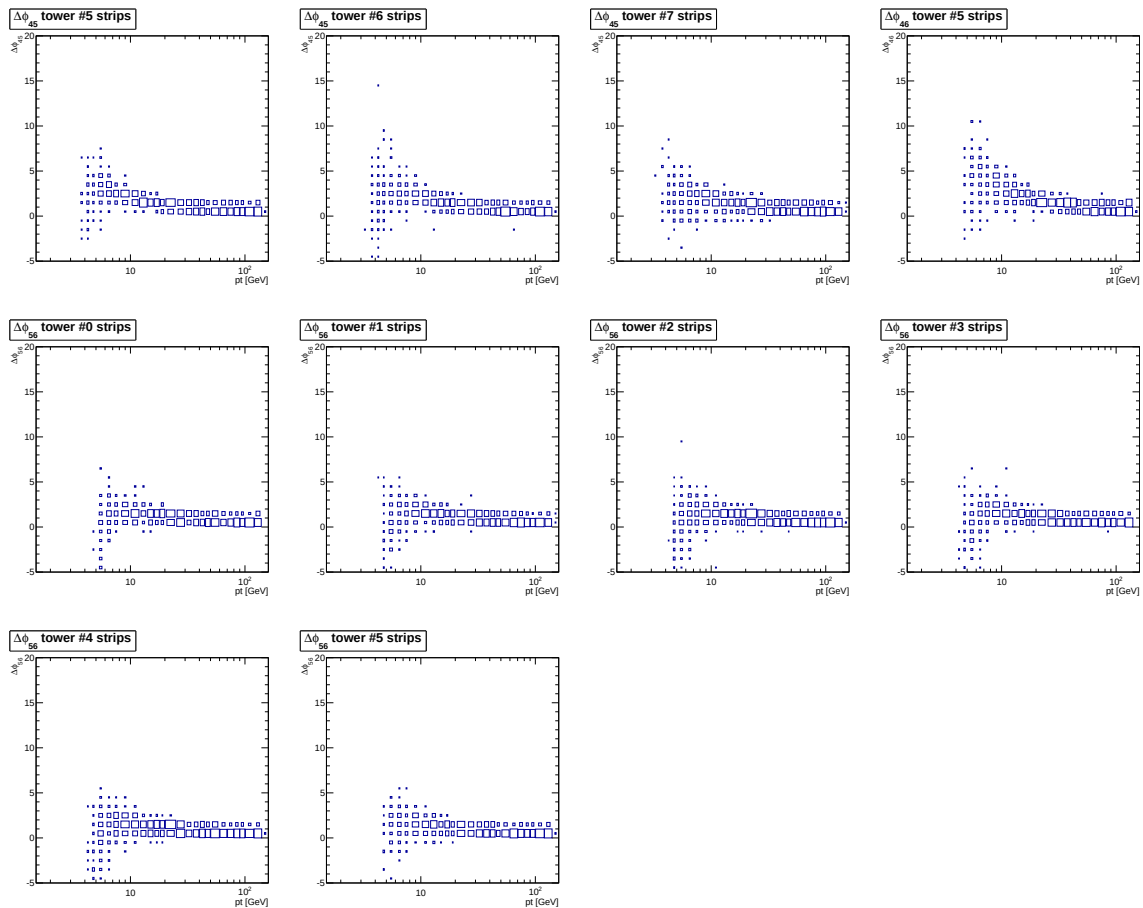
Wykresy zależności $\Delta\phi_{23}$ od pędu poprzecznego mionu dla wież numer 0 do 12, przyporządkowując kątowni ϕ odpowiedni numer paska.



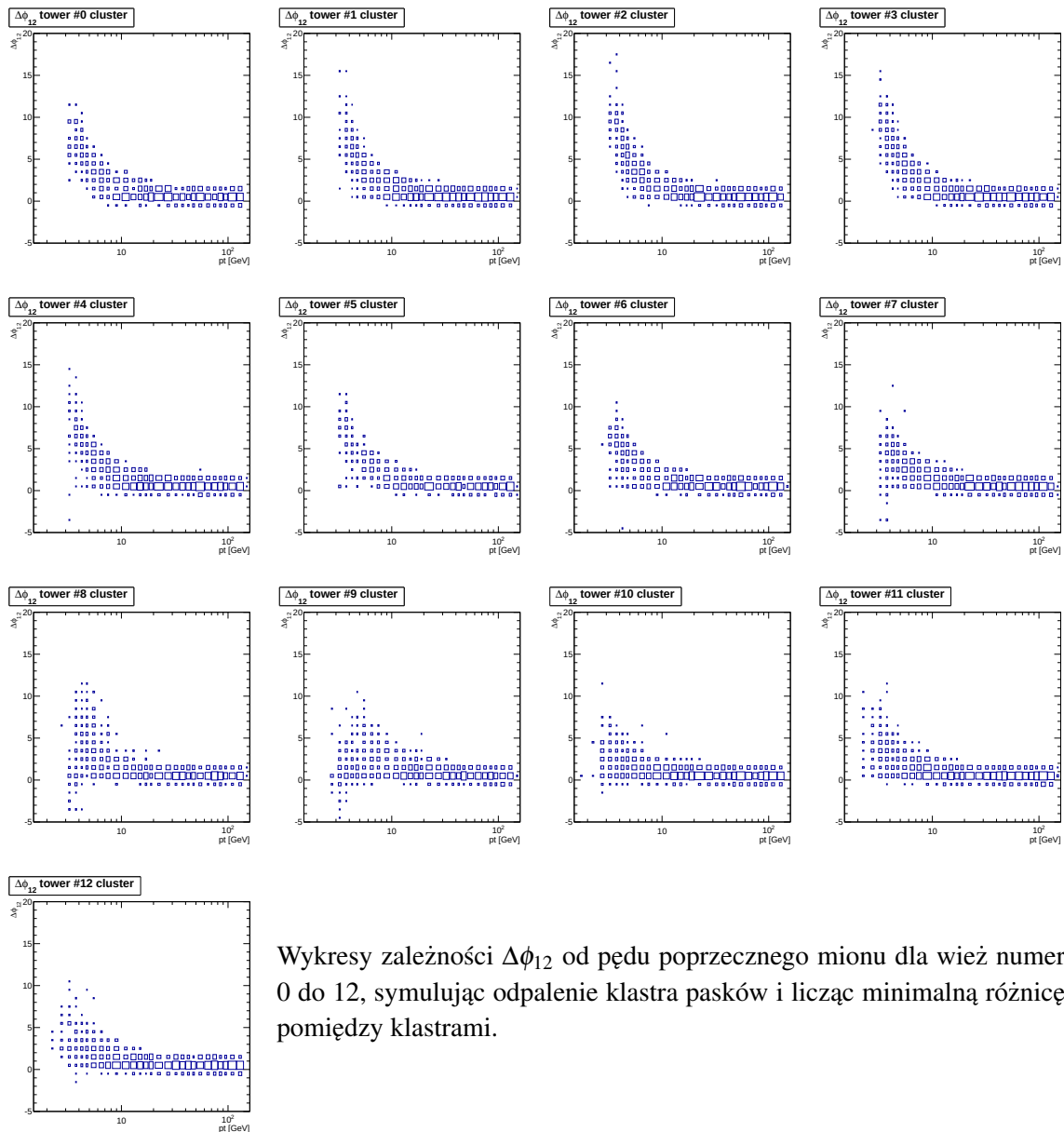
Wykresy zależności $\Delta\phi_{24}$ i $\Delta\phi_{34}$ od pędu poprzecznego mionu dla wież numer 0 do 8, przyporządkowując kątowi ϕ odpowiedni numer paska.



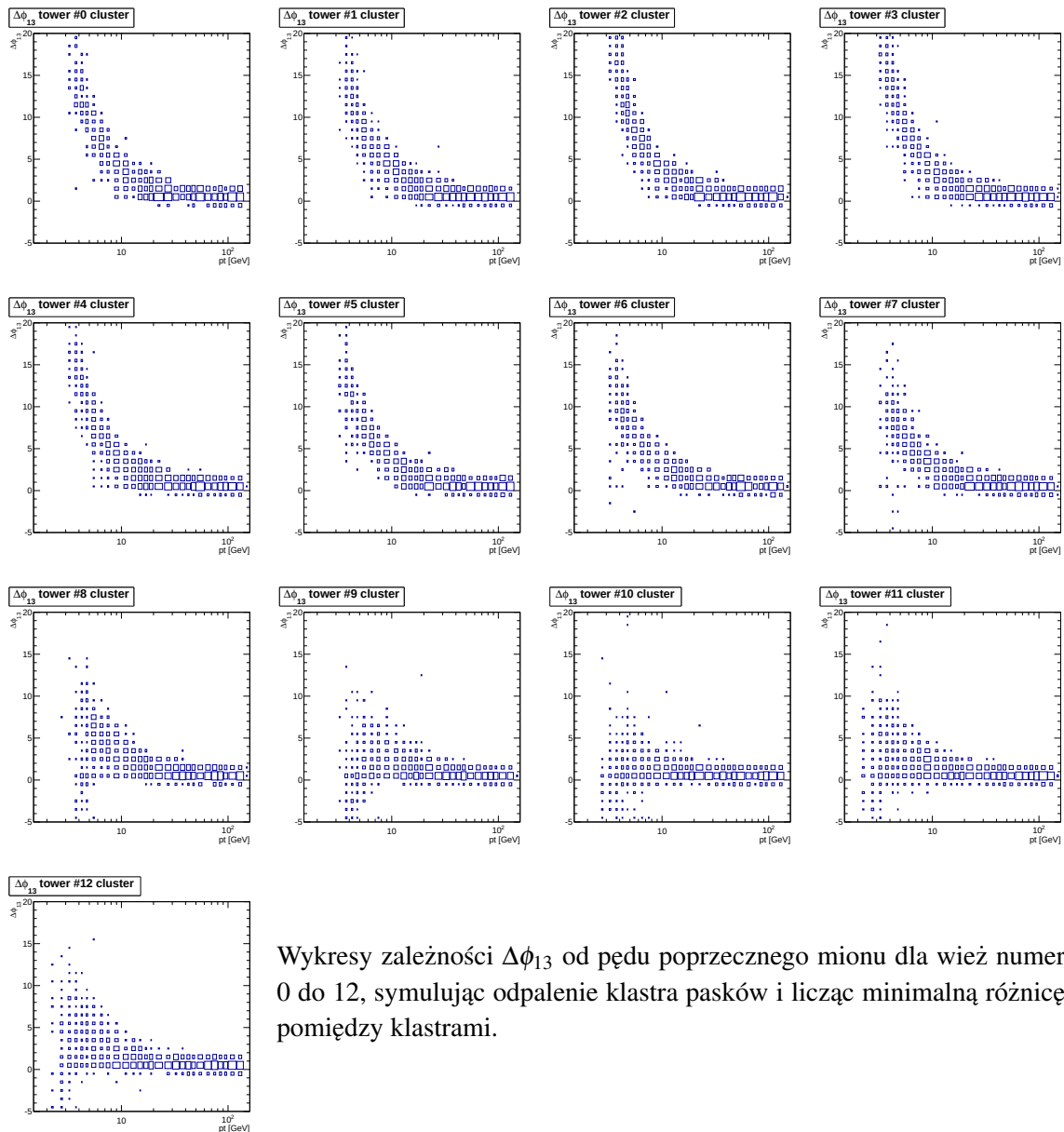
Wykresy zależności $\Delta\phi_{35}$ i $\Delta\phi_{36}$ od pędu poprzecznego mionu dla wież numer 0 do 4, przyporządkowując kątowi ϕ odpowiedni numer paska.



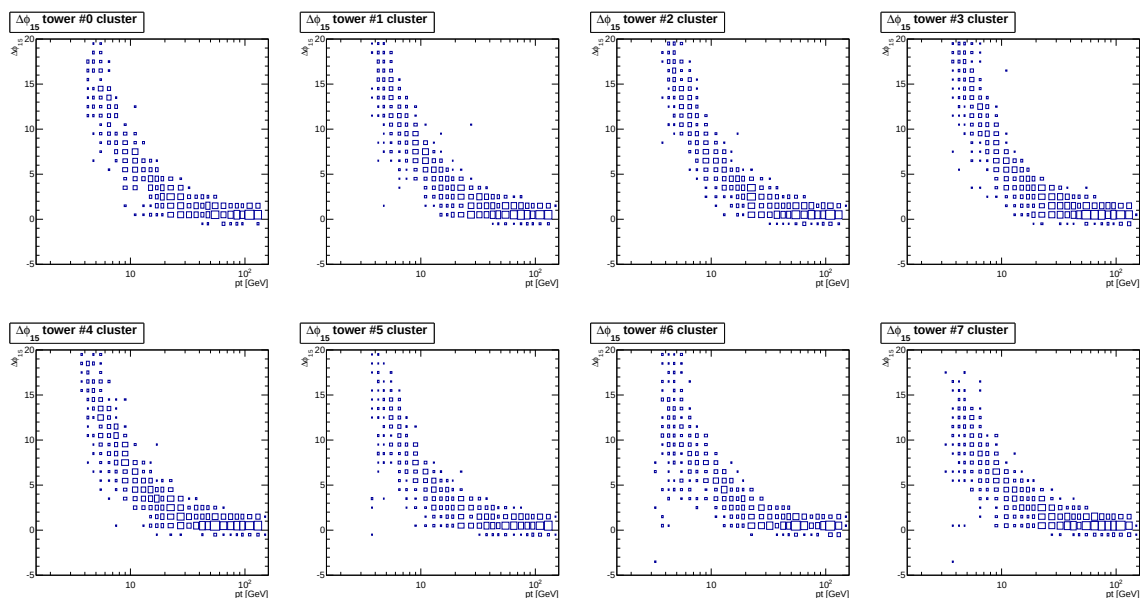
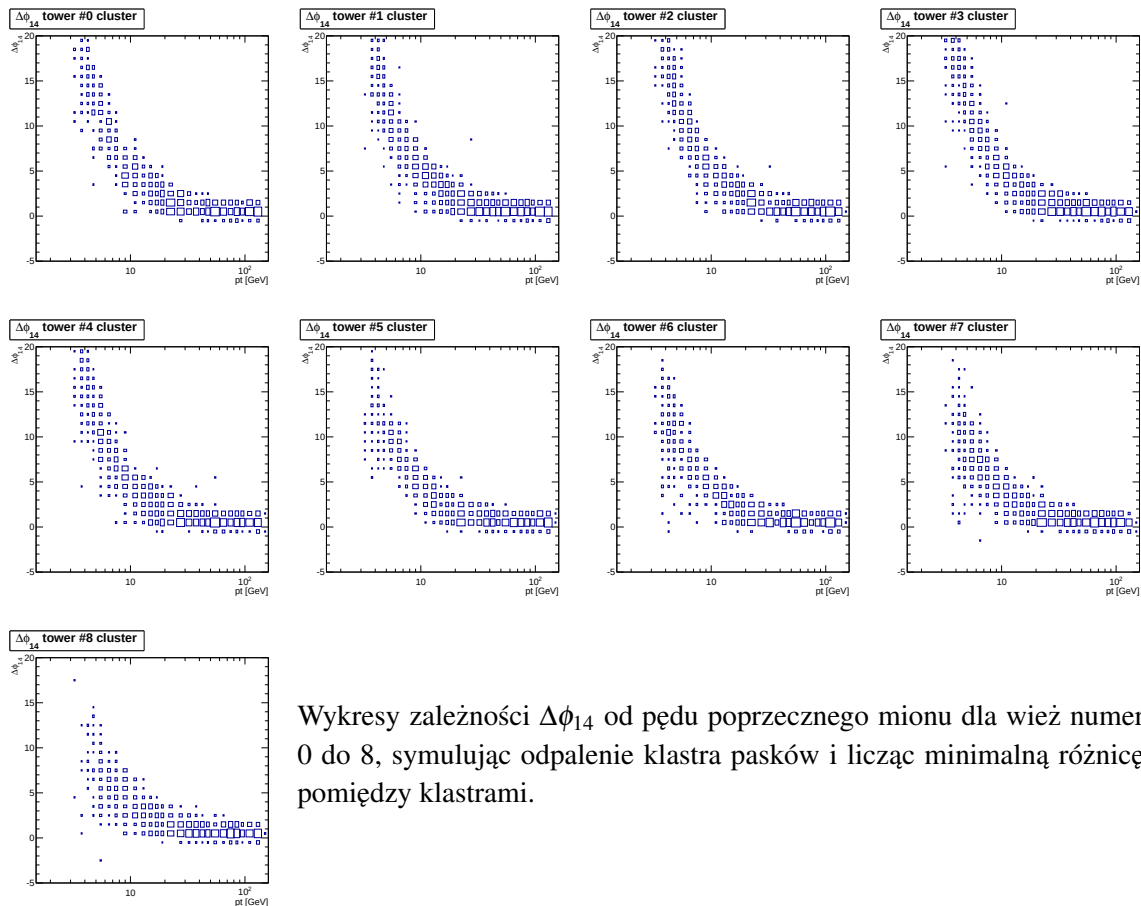
Wykresy zależności $\Delta\phi_{45}$, $\Delta\phi_{46}$ i $\Delta\phi_{56}$ od pędu poprzecznego mionu dla wień numer 0 do 7, przyporządkowując kątom ϕ odpowiedni numer paska.

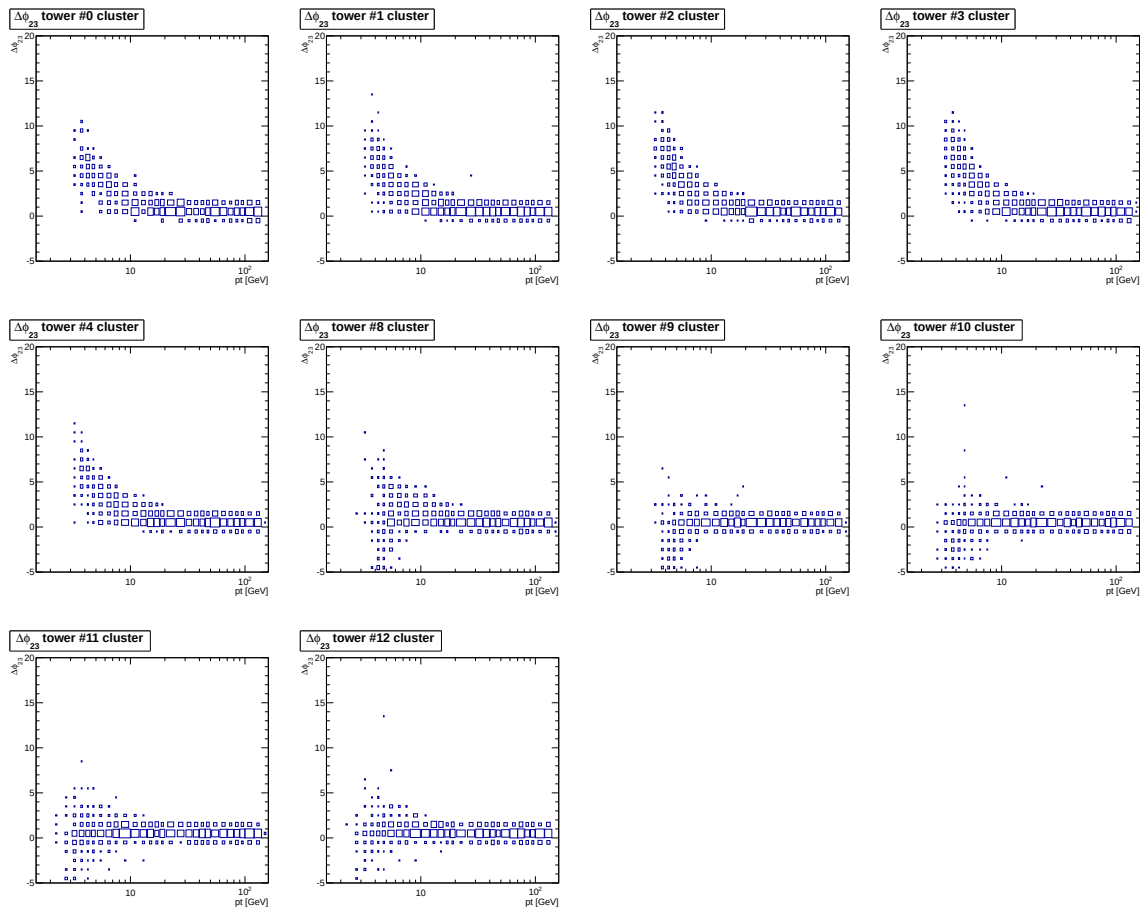


Wykresy zależności $\Delta\phi_{12}$ od pędu poprzecznego mionu dla wież numer 0 do 12, symulując odpalenie klastra pasków i licząc minimalną różnicę pomiędzy klastrami.

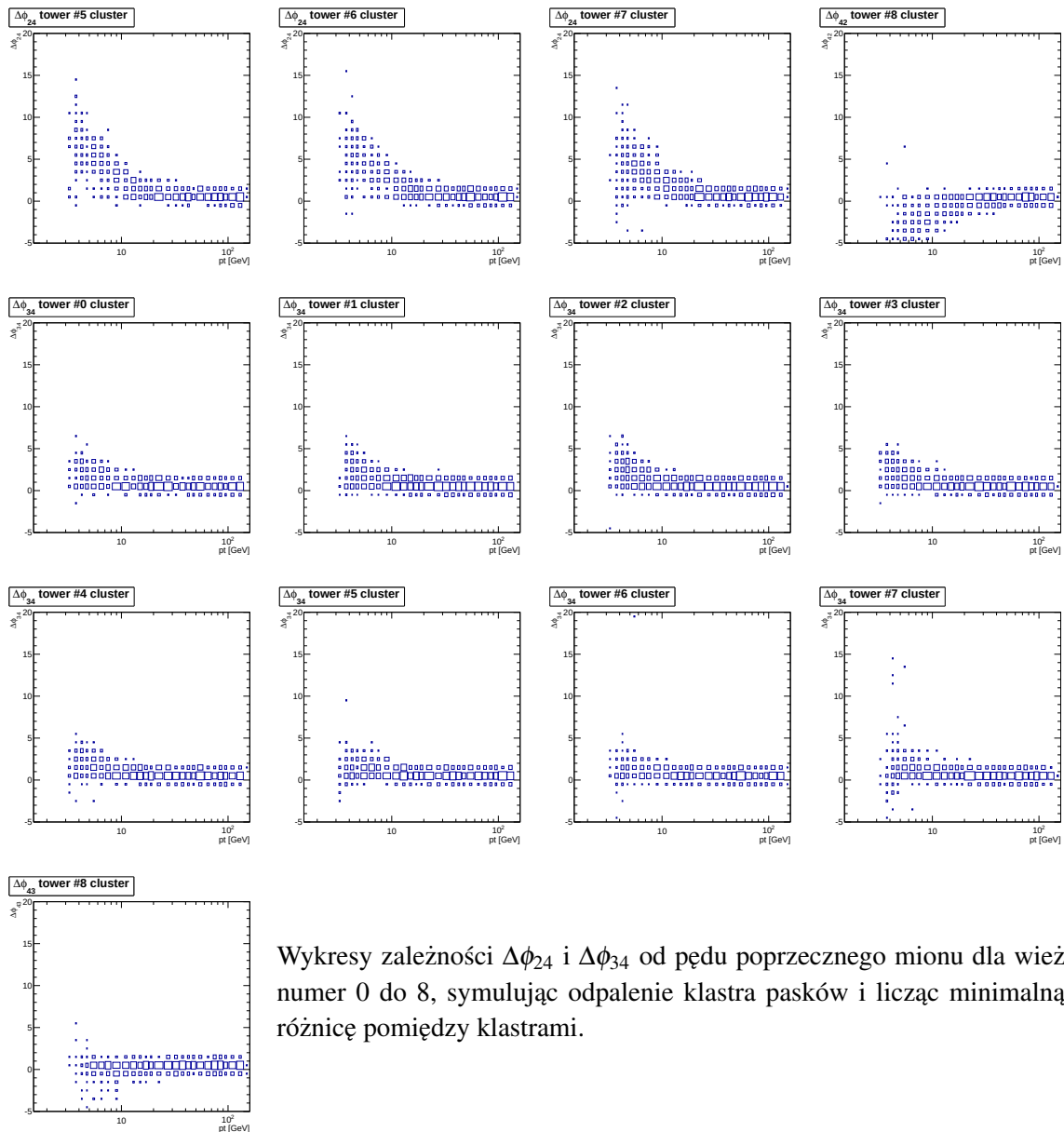


Wykresy zależności $\Delta\phi_{13}$ od pędu poprzecznego mionu dla wież numer 0 do 12, symulując odpalenie klastra pasków i licząc minimalną różnicę pomiędzy klastrami.

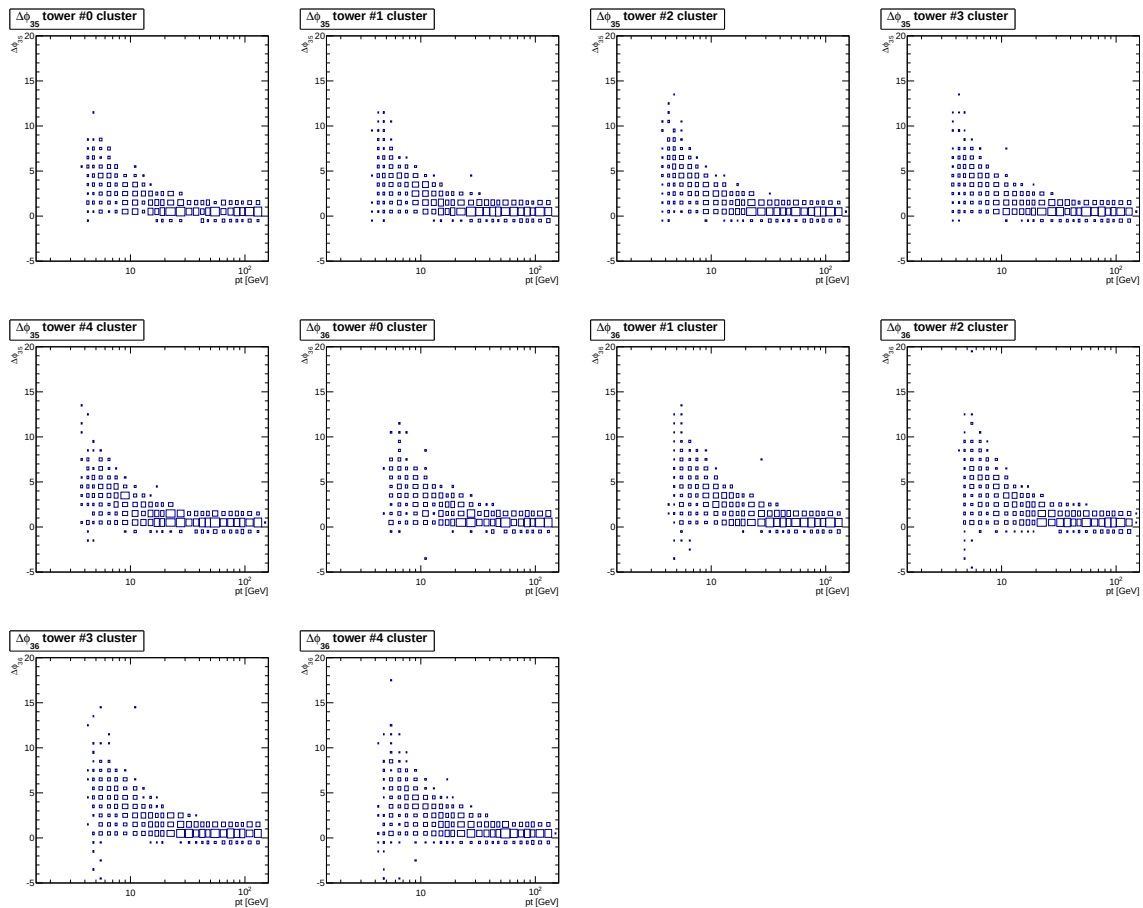




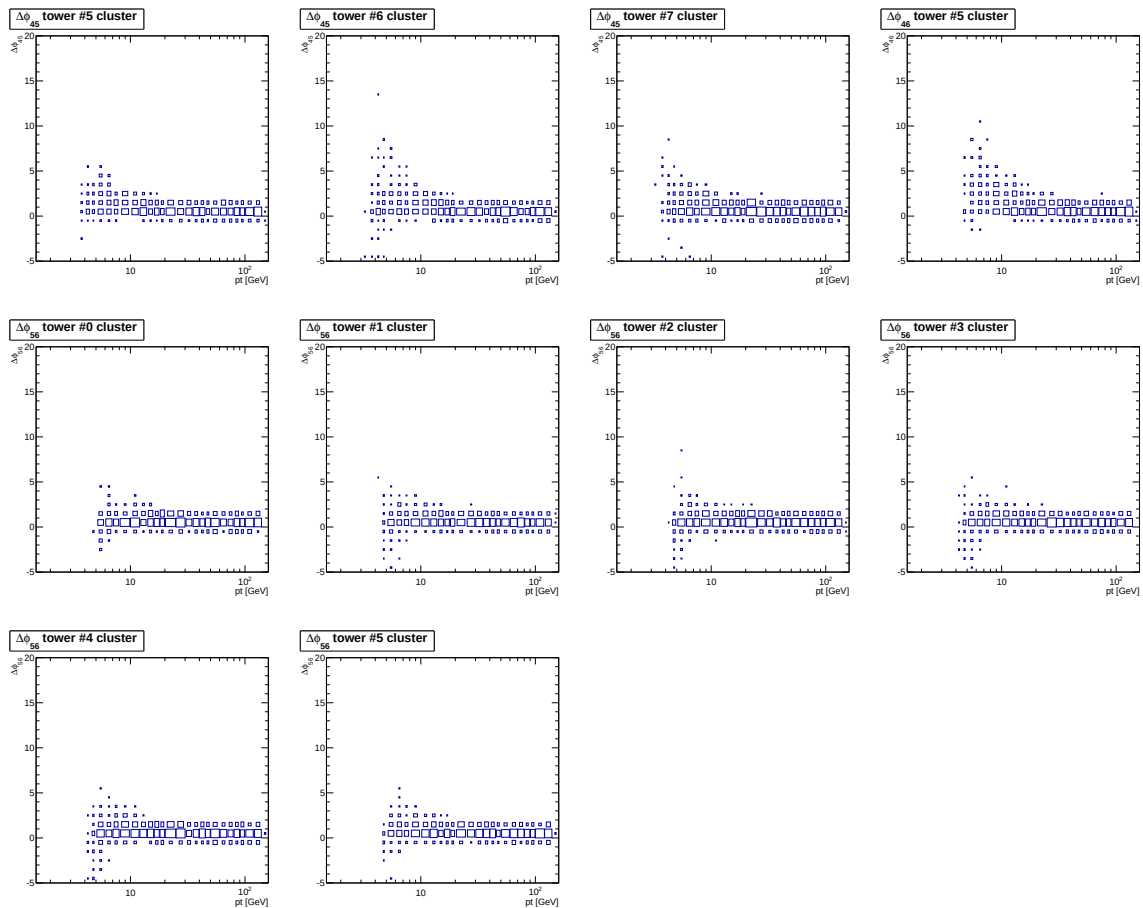
Wykresy zależności $\Delta\phi_{23}$ od pędu poprzecznego mionu dla wież numer 0 do 12, symulując odpalenie klastra pasków i licząc minimalną różnicę pomiędzy klastrami.



Wykresy zależności $\Delta\phi_{24}$ i $\Delta\phi_{34}$ od pędu poprzecznego mionu dla wież numer 0 do 8, symulując odpalenie klastra pasków i licząc minimalną różnicę pomiędzy klastrami.

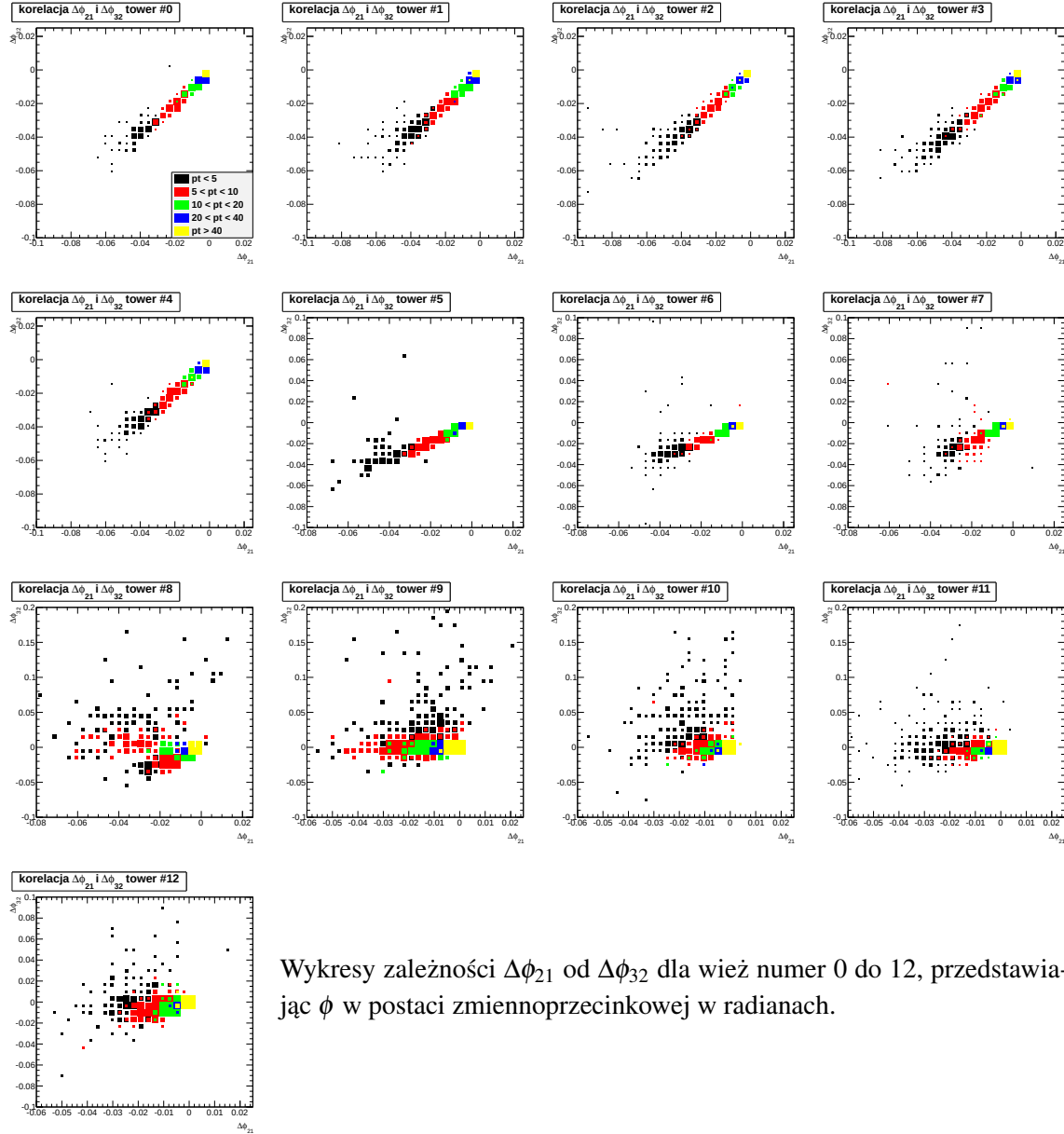


Wykresy zależności $\Delta\phi_{35}$ i $\Delta\phi_{36}$ od pędu poprzecznego mionu dla wież numer 0 do 4, symulując odpalenie klastra pasków i licząc minimalną różnicę pomiędzy klastrami.



Wykresy zależności $\Delta\phi_{45}$, $\Delta\phi_{46}$ i $\Delta\phi_{56}$ od pędu poprzecznego mionu dla wież numer 0 do 7, symulując odpalenie klastra pasków i licząc minimalną różnicę pomiędzy klastrami.

5 Dodatek: Wykresy korelacji $\Delta\phi_{21}$ i $\Delta\phi_{32}$



Wykresy zależności $\Delta\phi_{21}$ od $\Delta\phi_{32}$ dla wież numer 0 do 12, przedstawiając ϕ w postaci zmiennoprzecinkowej w radianach.

6 Dodatek: Kod procedury analizującej dane

```
#include "FWCore/Framework/interface/EDAnalyzer.h"
#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/EventSetup.h"
#include "DataFormats/Common/interface/Handle.h"
#include "FWCore/Framework/interface/ESHandle.h"
#include "FWCore/ParameterSet/interface/ParameterSet.h"
#include "FWCore/MessageLogger/interface/MessageLogger.h"

#include "SimDataFormats/TrackingHit/interface/PSimHitContainer.h"
#include "SimDataFormats/TrackingHit/interface/PSimHit.h"
```

```

#include "Geometry/CommonDetUnit/interface/GeomDet.h"
#include "DataFormats/MuonDetId/interface/RPCDetId.h"
#include "DataFormats/RPCRecHit/interface/RPCRecHitCollection.h"
#include "DataFormats/DetId/interface/DetIdCollection.h"
#include "DataFormats/TrackReco/interface/HitPattern.h"
#include "Geometry/RPCGeometry/interface/RPCGeometry.h"
#include "TrackingTools/Records/interface/TrackingComponentsRecord.h"
#include "Geometry/RPCGeometry/interface/RPCRoll.h"

#include "FWCore/Framework/interface/MakerMacros.h"
#include "Utilities.cc"
#include "RPCDetIdUtil.h"

#include "TProfile.h"
#include "TH1D.h"
#include "TH2D.h"
#include "TFile.h"
#include <string>
#include <sstream>
#include <cmath>

#include "DataFormats/Math/interface/deltaPhi.h"
#include "SimDataFormats/Track/interface/SimTrack.h"
#include "SimDataFormats/Track/interface/SimTrackContainer.h"

using namespace std;
template <class T> T sqr( T t ) {return t*t;}

class Cwiczenie : public edm::EDAnalyzer {
public:
    explicit Cwiczenie(const edm::ParameterSet& conf);
    ~Cwiczenie();
    virtual void beginJob();
    virtual void analyze(const edm::Event&, const edm::EventSetup&);
    virtual void endJob();
private:
    edm::ParameterSet theConfig;
    unsigned int theEventCount;
    L1PtScale theL1PtScale;
    L1RpcEtaScale theL1RpcEtaScale;
    TFile* theFile;
    TH2D* hZR;
    TH2D* hPhi;
    TH2D* hDeltaPhi;
    TH2D* hDeltaPhiAll;
    TH2D hLayersTest[6];
    TH2D hBarrelLayer[6];
    TH2D hEndcapLayer[3];
    typedef std::map<string, TH2D*> HistoMap;
    HistoMap theDeltaPhi;
    HistoMap theCorrelations;
};

Cwiczenie::Cwiczenie(const edm::ParameterSet& conf)
: theConfig(conf), theEventCount(0)
{

```

```

    edm::LogInfo("Cwiczenie")<<"_CTORXX";
}

Cwiczenie::~Cwiczenie()
{
    //delete theAnalysis;
    edm::LogInfo("Cwiczenie")<<"_DTOR";
}

void Cwiczenie::beginJob()
{
    std::cout << "Creating_file_" << theConfig.getParameter<std::string>("
        fileName").c_str() << ".root" << std::endl;
    string filename = "output_" + theConfig.getParameter<std::string>("
        fileName") + ".root";
    theFile = new TFile(filename.c_str(),"recreate");
    hZR = new TH2D("hZR","hZR",400,0,1200,400,0,800);
    hPhi = new TH2D("hPhi","hPhi",theL1PtScale.nPtBins,theL1PtScale.ptBins
        ,1152,-576,576);
    std::stringstream tmps;
    std::stringstream tmps2;
    std::stringstream tmps3;
    for(int i = 0; i<6; i++)
    {
        tmps << "hLayer_" << i+1;
        tmps2 << "hBarrelLayer_" << i+1;
        tmps3 << "hEndcapLayer_" << i+1;
        hLayersTest[i] = TH2D(tmps.str().c_str(), tmps.str().c_str()
            ,400,0,1200,400,0,800);
        hBarrelLayer[i] = TH2D(tmps2.str().c_str(), tmps2.str().c_str()
            ,400,0,1200,400,0,800);
        if(i<3) hEndcapLayer[i] = TH2D(tmps3.str().c_str(), tmps3.str().c_str
            (),400,0,1200,400,0,800);
        tmps.str(std::string());
        tmps2.str(std::string());
        tmps3.str(std::string());
    }
    char delta_phi04[19] = "121314152334353656";
    char delta_phi5[19] = "142434454612131556";
    char delta_phi67[15] = "14243445121315";
    char delta_phi8[13]= "121314424323";
    char delta_phi912[7]= "121323";

    string tmp, tmp2, tmp3, tmp4;
    double delta_phi_bins[26]; for(int i=0;i<26;i++) delta_phi_bins[i] = -5 +
        i;

    for(int i_tower = 0; i_tower<13; i_tower++)
    {
        for(int j = 0; j<9; j++)
        {
            stringstream ts;
            ts << i_tower << "_";
            if(i_tower<5) ts << delta_phi04[2*j] << delta_phi04[2*j+1];
            else if(i_tower==5) ts << delta_phi5[2*j] << delta_phi5[2*j+1];
            else if(i_tower==6 || i_tower ==7)

```

```

    {
        if(j<7) ts << delta_phi67[2*j] << delta_phi67[2*j+1];
        else continue;
    }
else if(i_tower==8)
{
    if(j<6) ts << delta_phi8[2*j] << delta_phi8[2*j+1];
    else continue;
}
else if(i_tower>8)
{
    if(j<3) ts << delta_phi912[2*j] << delta_phi912[2*j+1];
    else continue;
}
tmp = ts.str();
tmp2 = "hDeltaPhi_F_tower_" + tmp;
tmp3 = "hDeltaPhi_I_tower_" + tmp;
tmp4 = "hDeltaPhi_I_cluster_tower_" + tmp;
theDeltaPhi[tmp2] = new TH2D(tmp2.c_str(),tmp2.c_str(),
    theL1PtScale.nPtBins-2, theL1PtScale.ptBins+2,75,-5*TMath::Pi
    ()/576,20*TMath::Pi()/576);
theDeltaPhi[tmp3] = new TH2D(tmp3.c_str(),tmp3.c_str(),
    theL1PtScale.nPtBins-2, theL1PtScale.ptBins+2,25,
    delta_phi_bins);
theDeltaPhi[tmp4] = new TH2D(tmp4.c_str(),tmp4.c_str(),
    theL1PtScale.nPtBins-2, theL1PtScale.ptBins+2,25,
    delta_phi_bins);
std::cout << "Booking_histo:_" << tmp2 << std::endl;
}
}
string tmp5;
for(int i_tower = 0; i_tower<13; i_tower++)
{
    for(int i_pt = 0; i_pt < 5; i_pt++)
    {
        stringstream ts2;
        ts2 << i_tower << "_";
        if(i_pt == 0) ts2 << "5";
        if(i_pt == 1) ts2 << "10";
        if(i_pt == 2) ts2 << "20";
        if(i_pt == 3) ts2 << "40";
        if(i_pt == 4) ts2 << "40+";
        tmp5 = "hCorrelations_" + ts2.str();
        if(i_tower<5) theCorrelations[ts2.str()] = new TH2D(tmp5.c_str(),
            tmp5.c_str(),30,-0.1,0.025,30,-0.1,0.025);
        else if(i_tower > 4 && i_tower < 8) theCorrelations[ts2.str()] =
            new TH2D(tmp5.c_str(),tmp5.c_str(),30,-0.08,0.025,30,-0.1,0.1)
            ;
        else if(i_tower == 8) theCorrelations[ts2.str()] = new TH2D(tmp5.
            c_str(),tmp5.c_str(),30,-0.08,0.025,30,-0.1,0.2);
        else if(i_tower > 8 && i_tower < 12) theCorrelations[ts2.str()] =
            new TH2D(tmp5.c_str(),tmp5.c_str(),
            30,-0.06,0.025,30,-0.1,0.2);
        else if(i_tower == 12) theCorrelations[ts2.str()] = new TH2D(tmp5
            .c_str(),tmp5.c_str(),30,-0.06,0.025,30,-0.1,0.1);
        std::cout << "Booking_histo:_" << ts2.str() << std::endl;
        ts2.str(std::string());
    }
}

```

```

    }
}

void Cwiczenie::endJob() {
    std::cout << "Saving file ..." << std::endl;
    hZR->Write();
    hPhi->Write();
    std::map<string, TH2D*>::iterator iter;
    for(iter = theDeltaPhi.begin(); iter != theDeltaPhi.end(); iter++)
    {
        iter->second->Write();
    }
    for(iter = theCorrelations.begin(); iter != theCorrelations.end(); iter++)
    {
        iter->second->Write();
    }

    for(int i = 0; i<6; i++) hLayersTest[i].Write();
    for(int i = 0; i<6; i++) hBarrelLayer[i].Write();
    for(int i = 0; i<3; i++) hEndcapLayer[i].Write();
    theFile->Close();
}

int deltaCluster(pair<int, int>p1, pair<int, int>p2)
{
    int delta = -1;
    pair<int, int> gp1;
    pair<int, int> gp2;
    //checking if the pairs are sorted
    if(p1.first < p1.second) gp1 = p1;
    else
    {
        gp1.first = p1.second;
        gp1.second = p1.first;
    }
    if(p2.first < p2.second) gp2 = p2;
    else
    {
        gp2.first = p2.second;
        gp2.second = p2.first;
    }
    //gp1 gp2 are now good (sorted) pairs
    if(gp2.first >= gp1.first && gp2.first <= gp1.second)
        delta = 0;
    else if(gp2.first > gp1.first && gp2.first > gp1.second)
        delta = gp1.second - gp2.first;
    else if(gp2.second >= gp1.first && gp2.second <= gp1.second)
        delta = 0;
    else if(gp2.second < gp1.first && gp2.second < gp1.second)
        delta = gp1.first - gp2.second;
    return delta;
}

void deltaPhiFill(std::map<std::string, TH2D*>& DeltaPhi, std::map<std::string, double>& f, std::map<std::string, int>& i, std::map<std::string, std::pair<int, int>>& ic, double pt, int charge, char xx, char yy, int

```

```

tower) // fill histogram for deltaphi_xy
{
    stringstream key;
    key << xx;
    string x = key.str();
    key.str(std::string());
    key << yy;
    string y = key.str();
    key.str(std::string());
    key << tower << "_";
    if(f.find(key.str()+x) != f.end() && f.find(key.str()+y) != f.end())
    {
        DeltaPhi["hDeltaPhi_F_tower_"+key.str()+x+y]->Fill(pt, charge*
            deltaPhi(f[key.str()+y], f[key.str()+x]));
        std::cout << "Filling_histo:_" << "hDeltaPhi_F_tower_"+key.str()+x+y
            << std::endl;
    }
    if(i.find(key.str()+x) != i.end() && i.find(key.str()+y) != i.end())
    {
        DeltaPhi["hDeltaPhi_I_tower_"+key.str()+x+y]->Fill(pt, charge*(i[
            key.str()+y] - i[key.str()+x]));
    }
    if(ic.find(key.str()+x) != ic.end() && ic.find(key.str()+y) != ic.end
        ())
    {
        DeltaPhi["hDeltaPhi_I_cluster_tower_"+key.str()+x+y]->Fill(pt,
            charge*(deltaCluster(ic[key.str()+y], ic[key.str()+x])));
    }
}

void Cwiczenie::analyze(
    const edm::Event& ev, const edm::EventSetup& es)
{
    edm::Handle< edm::PSimHitContainer > simHits;
    ev.getByLabel("g4SimHits", "MuonRPCHits", simHits);
    edm::Handle< edm::SimTrackContainer> simTk;
    ev.getByLabel("g4SimHits", simTk);
    std::vector<SimTrack> mySimTracks = *(simTk.product());
    double pt = -1000;
    double eta = -1000;
    int charge = 0;
    for(std::vector<SimTrack>::const_iterator it=mySimTracks.begin(); it<
        mySimTracks.end(); it++)
    {
        const SimTrack & track = *it;
        if(track.type() == 13 || track.type() == -13)
        {
            if(track.type() > 0) charge = 1;
            else charge = -1;
            pt = track.momentum().pt();
            eta = track.momentum().eta();
            break;
        }
    }

    edm::ESHandle<RPCGeometry> rpcGeometry;
    es.get<MuonGeometryRecord>().get(rpcGeometry);
    std::vector<RPCRoll*> rpcRolls = rpcGeometry->rolls();

```

```

double r;
double z;
double phi;
int phi_strip;
double eta_det;
int layer;
int barrel_layer;
int endcap_layer;
int tower = -1;
int clustersize = -1;
const double aveCls = 1.7;
stringstream key;
typedef std::map<string , double> FMap;
typedef std::map<string , int> IMap;
typedef std::map<string , std::pair<int , int>> ICMMap;
FMap PhiF;
IMap PhiI;
ICMMap PhiIC;
for(int i = 1; i < 16; i++)
{
    if(abs(eta) < theL1RpcEtaScale.etaBins[17])
    {
        tower = 0;
        break;
    }
    if(abs(eta) > theL1RpcEtaScale.etaBins[i+16] && abs(eta) <
        theL1RpcEtaScale.etaBins[i+17])
    {
        tower = i;
        break;
    }
}
std::cout << "eta_=" << eta << "_tower_=" << tower << std::endl;

for (edm::PSimHitContainer::const_iterator hitItr = simHits->begin();
    hitItr != simHits->end(); ++hitItr) {
    RPCDetId rpcDetId = hitItr->detUnitId();
    RPCDetIdUtil rpcDetIdUtil = RPCDetIdUtil(rpcDetId);
    if ( !(13 == abs(hitItr->particleType()) ) ) continue;
    GlobalPoint detPosition = rpcGeometry->idToDet(rpcDetId)->position();
    GlobalPoint hitPosition = rpcGeometry->idToDet(rpcDetId)->toGlobal(
        hitItr->localPosition());

    r = hitPosition.perp();
    z = hitPosition.z();
    phi = hitPosition.phi();
    eta_det = detPosition.eta();
    phi_strip = -999;

    hZR->Fill(abs(z),r);
    hPhi->Fill(pt,phi);
    layer = rpcDetIdUtil.layer(eta_det);
    barrel_layer = rpcDetIdUtil.barrelLayer();
    endcap_layer = rpcDetIdUtil.endcapLayer();
    if(layer<7 && layer > 0) hLayersTest[layer-1].Fill(abs(z),r);
    if(barrel_layer > 0 && barrel_layer < 7 && abs(z) < 675) hBarrelLayer[
        barrel_layer-1].Fill(abs(z),r);
}

```

```

    if(endcap_layer > 0 && endcap_layer < 4 && abs(z) > 675) hEndcapLayer[
        endcap_layer-1].Fill(abs(z),r);
    std::cout << "layer_" << layer << std::endl;
    std::cout << "pt_" << pt << std::endl;
    std::cout << "charge_" << charge << std::endl;

    if(layer < 1 || tower < 0) continue;
    key << tower << "_" << layer;

    PhiF[key.str()] = phi;
    if(phi >= -TMath::Pi() && phi <= TMath::Pi())
    {
        phi_strip = static_cast<int>(ceil(phi*576/TMath::Pi()));
    }

    if(phi_strip < -576 || phi_strip > 576) std::cout << "phi_strip_error"
        << std::endl;
    PhiI[key.str()] = phi_strip;

    //cluster size:
    double w = rand()/double(RAND_MAX);
    if (w< 1.e-10) w = 1.e-10;
    clustersize = static_cast<int>(-1.*aveCls*log(w)+1.);
    if(phi_strip > -576 && phi_strip < 576 && clustersize > 0)
    {
        if(clustersize == 1) PhiIC[key.str()] = pair<int,int>(phi_strip,
            phi_strip);
        else if(clustersize%2 != 0) PhiIC[key.str()] = pair<int,int>(
            phi_strip-(clustersize-1)/2,phi_strip+(clustersize-1)/2);
        else if(clustersize%2 == 0)
        {
            if(rand()/double(RAND_MAX) < 0.5)
                PhiIC[key.str()] = pair<int,int>(phi_strip-clustersize/2,
                    phi_strip+clustersize/2-1);
            else
                PhiIC[key.str()] = pair<int,int>(phi_strip-clustersize/2-1,
                    phi_strip+clustersize/2);
        }
    }
    key.str(std::string());
}

key.str(std::string());

char delta_phi04[19] = "121314152334353656";
char delta_phi5[19] = "142434454612131556";
char delta_phi67[15] = "14243445121315";
char delta_phi8[13] = "121314424323";
char delta_phi912[7] = "121323";

for(int i = 0; i < 5; i++)
    for(int j = 0; j < 9; j++)
        deltaPhiFill(theDeltaPhi,PhiF,PhiI,PhiIC,pt,charge,delta_phi04[2*
            j],delta_phi04[2*j+1],i);
for(int i = 5; i < 6; i++)
    for(int j = 0; j < 9; j++)
        deltaPhiFill(theDeltaPhi,PhiF,PhiI,PhiIC,pt,charge,delta_phi5[2*j
            ],delta_phi5[2*j+1],i);

```

```

for(int i = 6; i < 8; i++)
    for(int j = 0; j < 7; j++)
        deltaPhiFill(theDeltaPhi, PhiF, PhiI, PhiIC, pt, charge, delta_phi67[2*
            j], delta_phi67[2*j+1], i);
for(int i = 8; i < 9; i++)
    for(int j = 0; j < 6; j++)
        deltaPhiFill(theDeltaPhi, PhiF, PhiI, PhiIC, pt, charge, delta_phi8[2*j
            ], delta_phi8[2*j+1], i);
for(int i = 9; i < 13; i++)
    for(int j = 0; j < 3; j++)
        deltaPhiFill(theDeltaPhi, PhiF, PhiI, PhiIC, pt, charge, delta_phi912
            [2*j], delta_phi912[2*j+1], i);

for(int i_tower = 0; i_tower < 13; i_tower++)
{
    stringstream ckey;
    stringstream xkey;
    stringstream ykey;
    ckey << i_tower << "_";
    if(pt < 5) ckey << "5";
    if(pt > 5 && pt < 10) ckey << "10";
    if(pt > 10 && pt < 20) ckey << "20";
    if(pt > 20 && pt < 40) ckey << "40";
    if(pt > 40) ckey << "40+";
    xkey << i_tower << "_";
    ykey << i_tower << "_";
    double bin1 = 0.0001;
    if( !(charge*deltaPhi(PhiF[ykey.str()+"1"], PhiF[
        xkey.str()+"2"]) < bin1 && charge*deltaPhi(PhiF[
        ykey.str()+"1"], PhiF[xkey.str()+"2"]) > -bin1)
        && !(charge*deltaPhi(PhiF[ykey.str()+"2"], PhiF[
        xkey.str()+"3"]) < bin1 && charge*deltaPhi(PhiF[
        ykey.str()+"2"], PhiF[xkey.str()+"3"]) > -bin1)
        && PhiF[ykey.str()+"1"] != 0 && PhiF[ykey.str()+
        "2"] != 0 && PhiF[xkey.str()+"3"] != 0)
        theCorrelations[ckey.str()]->Fill(charge*deltaPhi(PhiF[ykey.str()+"1"
            ], PhiF[xkey.str()+"2"]), charge*deltaPhi(PhiF[ykey.str()+"2"], PhiF[
            xkey.str()+"3"]));
    ckey.str(std::string());
    xkey.str(std::string());
    ykey.str(std::string());
}
}

DEFINE_FWK_MODULE(Cwiczenie);

```