KaonLTMeeting

August 10th, 2023

Richard Trotta

LTSep Analysis Procedure

1. All analysis cuts per run (i.e. PID, acceptance, timing) are applied.
2. Diamond cuts are drawn on the CENTER setting.
3. Apply random subtraction to data and dummy.
4. Combine all settings and choose t/phi bins for low ε. These bins will also be used of high ε, so check high ε as well.
5. Compare SIMC to data/dummy setting by setting.
6. Combine settings again at for each Q^2, W, t-bin, φ-bin, ε for data, dummy, and SIMC.
    a. Dummy is subtracted from data bin by bin.
    b. The yield is calculated using the effective charge from data and normfactor/nevents is applied to normalize simc.
    c. The data and SIMC yields are compared and the R value per bin is obtained.
7. Calculate error weighted average of data and SIMC.

TH1F
1000 bin
histograms

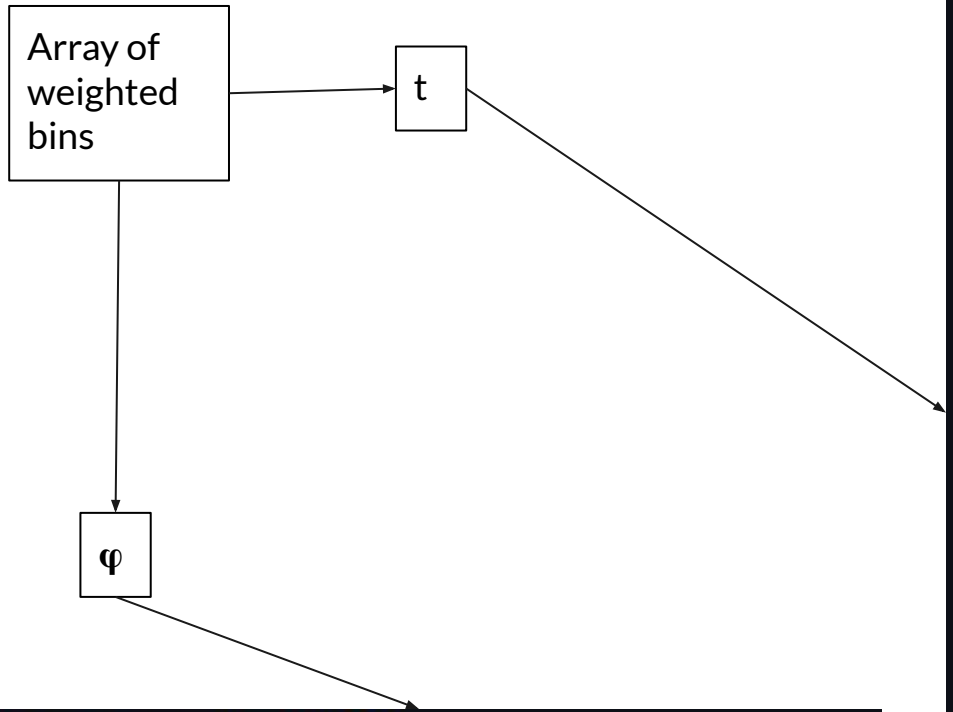
t
φ

```python
def weight_bins(histogram):

    # Get the number of bins in the histogram
    n_bins = histogram.GetNbinsX()

    # Calculate the integral for each bin and store it along with the bin edges
    bin_edges = []
    bin_integrals = []

    for i in range(1, n_bins + 1):
        bin_low_edge = histogram.GetXaxis().GetBinLowEdge(i)
        bin_integral = histogram.Integral(1, i)
        bin_edges.append(bin_low_edge)
        bin_integrals.append(bin_integral)

    # The last bin edge is the upper edge of the last bin
    bin_edges.append(histogram.GetXaxis().GetBinUpEdge(n_bins))

    # Calculate the total integral of the histogram (integral up to the last bin)
    total_integral = histogram.Integral()

    # Calculate the weights for each bin based on their integrals
    bin_weights = [integral / total_integral for integral in bin_integrals]

    # Weight the bin edges by the bin weights
    weighted_bin_edges = [edge * weight for edge, weight in zip(bin_edges, bin_weights)]

    return weighted_bin_edges
```

Arrays of bin_edges and integrated bins

Find bin weights

Apply bin weights to bin edges




```python
def find_tbins(H_t_BinTest):



    ##################################################################################################

    def histedges_equalN(x, nbin):
        # Grab number of events in array
        npt = len(x)
        # One-dimensional linear interpolation for monotonically increasing sample points.
        # Returns the one-dimensional piecewise linear interpolant to a function with given
        # discrete data points (xp, fp), evaluated at x.
        #
        # np.interp(x, xp, fp)
        # x -> np.linspace(0, npt, nbin + 1) : The x-coordinates at which to evaluate the interpol
        # In this case, this is an array of evenly spaced t-bins
        # xp -> np.arange(npt) : The x-coordinates of the data points
        # In this case, this returns evenly spaced values within a given interval
        # yp -> np.sort(x) : he y-coordinates of the data points
        # In this case, this returns a sorted copy of the array
        return np.interp(np.linspace(0, npt, nbin + 1),np.arange(npt),np.sort(x))

    print("\nFinding t bins...")
    # Histogram takes the array data set and the bins as input
    # The bins are determined by a linear interpolation (see function above)
    # This returns the binned data with equal number of events per bin
    # Returns...
    # n -> The values of the histogram bins
    # bins -> The edges of the bins
    # patches -> Container of individual artists used to create the histogram or list of
    # such containers if there are multiple input datasets.
    n, bins, patches = plt.hist(H_t_BinTest, histedges_equalN(H_t_BinTest, inpDict["NumtBins"]))

    print("t_bins = ", bins)
```

```python
def find_phibins(H_phi_BinTest):

    print("\nFinding phi bins...")
    phi_arr = np.linspace(0.0, 360.0, inpDict["NumPhiBins"]+1)

    n, bins, patches = plt.hist(H_phi_BinTest, phi_arr)

    print("phi_bins = ", bins)
```

TH1F
1000 bin
histograms

Q^2
W
MM

```python
def weight_bins(histogram):

    # Get the number of bins in the histogram
    n_bins = histogram.GetNbinsX()

    # Calculate the integral for each bin and store it along with the bin edges
    bin_edges = []
    bin_integrals = []

    for i in range(1, n_bins + 1):
        bin_low_edge = histogram.GetXaxis().GetBinLowEdge(i)
        bin_integral = histogram.Integral(1, i)
        bin_edges.append(bin_low_edge)
        bin_integrals.append(bin_integral)

    # The last bin edge is the upper edge of the last bin
    bin_edges.append(histogram.GetXaxis().GetBinUpEdge(n_bins))

    # Calculate the total integral of the histogram (integral up to the last bin)
    total_integral = histogram.Integral()

    # Calculate the weights for each bin based on their integrals
    bin_weights = [integral / total_integral for integral in bin_integrals]

    # Weight the bin edges by the bin weights
    weighted_bin_edges = [edge * weight for edge, weight in zip(bin_edges, bin_weights)]

    return weighted_bin_edges
```

Arrays of bin_edges and integrated bins

Find bin weights

Apply bin weights to bin edges

Array of weighted bins

t
φ
Q^2
W
MM

```
aver_lst = []
for j in range(len(t_bins) - 1):
    tbin_indices = np.where((float(t_bins[j]) <= t) & (t < float(t_bins[j + 1])))[0]
    tbin_index = j
    Q2_val = Q2[tbin_indices]
    W_val = W[tbin_indices]
    t_val = t[tbin_indices]
    # Append tbin_index, Q2, W, and t to aver_lst
    for k in range(len(phi_bins) - 1):
        phibin_indices = np.where((float(phi_bins[k]) <= phi_deg) & (phi_deg < float(phi_bins[k + 1])))[0]
        phibin_index = k
        # t binning
        #MM_val = MM[tbin_indices]
        # t+phi binning
        # Combine tbin_indices and phibin_indices using logical AND
        combined_indices = np.intersect1d(tbin_indices, phibin_indices)
        MM_val = MM[combined_indices]
        print("_____",tbin_index, phibin_index, len(MM), len(Q2), len(W), len(t),"_____")
        print("----------",tbin_index, phibin_index, len(MM_val), len(Q2_val), len(W_val), len(t_val),"----------")
        aver_lst.append((tbin_index, phibin_index, Q2_val, W_val, t_val, MM_val))
        print("_____",aver_lst,"_____\n")
```

Find t within tbins

Find Q2, W within tbins

Find φ within φ bins

Find MM within t/φ bins

Per bin….

```
# Calculate averages for each Q2, W, t value per t-bin
Q2_aver = [np.average(tup[2])]
W_aver = [np.average(tup[3])]
t_aver = [np.average(tup[4])]
# Find the number of events per t/phi bin
try:
    nevents = integrate.simps(tup[5])
except IndexError:
    nevents = 0
groups[key] = {
    "t_bins" : t_bins,
    "phi_bins" : phi_bins,
    "Q2_arr" : tup[2],
    "W_arr" : tup[3],
    "t_arr" : tup[4],
    "MM_arr" : tup[5],
    "Q2_aver" : Q2_aver,
    "W_aver" : W_aver,
    "t_aver" : t_aver,
    "nevents" : nevents
}
```

Find average Q2, W, t

Integrate over MM for number of events

Per bin….

```
# Calculate averages for each Q2, W, t value per t-bin
Q2_aver = [np.average(tup[2])]
W_aver = [np.average(tup[3])]
t_aver = [np.average(tup[4])]
# Find the number of events per t/phi bin
try:
    nevents = integrate.simps(tup[5])
except IndexError:
    nevents = 0
groups[key] = {
    "t_bins" : t_bins,
    "phi_bins" : phi_bins,
    "Q2_arr" : tup[2],
    "W_arr" : tup[3],
    "t_arr" : tup[4],
    "MM_arr" : tup[5],
    "Q2_aver" : Q2_aver,
    "W_aver" : W_aver,
    "t_aver" : t_aver,
    "nevents" : nevents
}
```

```
simc_nested_dict["yield_simc_{}".format(hist["phi_setting"])] = simc_nested_dict["nevents"]/(100*hist["normfac_simc"])
#################
#################
#################

print("{} Simc-> Tuple: {}, Simc yield: {}, ".format(hist["phi_setting"],simc_key_tuple,simc_nested_dict["yield_simc_{}".format(hist["phi_setting"])]))
# Subtract dummy from data per t/phi bin and get data yield
data_nested_dict["yield_data_{}".format(hist["phi_setting"])] = data_nested_dict["nevents"]*hist["normfac_data"] \
                                                              - dummy_nested_dict["nevents"]*hist["normfac_dummy"]
try:
    data_nested_dict["ratio_{}".format(hist["phi_setting"])] = \
                                                          data_nested_dict["yield_data_{}".format(hist["phi_setting"])] \
                                                          / simc_nested_dict["yield_simc_{}".format(hist["phi_setting"])]
except ZeroDivisionError:
    data_nested_dict["ratio_{}".format(hist["phi_setting"])] = 0
data_nested_dict["ratio_{}".format(hist["phi_setting"])] = np.where( \
                                                               np.isinf(data_nested_dict["ratio_{}".format(hist["phi_setting"])]), \
                                                               0, data_nested_dict["ratio_{}".format(hist["phi_setting"])])
print("{}-> Tuple: {}, Ratio: {}, ".format(hist["phi_setting"],data_key_tuple,data_nested_dict["ratio_{}".format(hist["phi_setting"])]))
```

SIMC Yield

Data yield w/ dummy subtracted

Ratio of Data/ SIMC