Normalisation integrals

- **So far we have assumed that every function is normalised internally. In the case of polynomials, this assumption is a little strained.**
- **We need a more general approach. Make some changes to `GeneralFcn` (full code in `example4.cu`):**

```
__constant__ double dev_norm_integrals[512];
double host_norm_integrals[512];

__device__ double dev_Gaussian (double xval, unsigned int pIdx) {
  double mean  = dev_params[dev_indices[pIdx + 1]];
  double sigma = dev_params[dev_indices[pIdx + 2]];
  return exp(-0.5*pow((xval - mean) / sigma, 2)); // External normalisation
}

// Similar change to other functions.
```

```
__device__ double callFunction (double xval,
                                unsigned int fcnIdx,
                                unsigned int parIdx) {
  dev_fcn_ptr theFunction;
  theFunction = reinterpret_cast<dev_fcn_ptr>(dev_fcn_table[fcnIdx]);
  double pdfVal = (*theFunction)(xval, parIdx);
  pdfVal *= dev_norm_integrals[parIdx];
  return pdfVal;
}

struct GeneralFcn : public thrust::unary_function<double, double> {
  GeneralFcn (unsigned int idx, unsigned int pid, bool tl = true)
    : fcnIdx(idx)
    , parIdx(pid)
    , takeLog(tl)
  {}

  __device__ double operator () (double xval) {
    double pdfVal = callFunction(xval, fcnIdx, parIdx);
    if (takeLog) return -2*log(pdfVal);
    else return pdfVal;
  }
private:
  unsigned int fcnIdx;
  unsigned int parIdx;
  bool takeLog;
};
```

```
void fcn_glue (int& npar, double* deriv, double& fVal,
               double param[], int flag) {
  cudaMemcpyToSymbol(dev_params, param, npar*sizeof(double));
  double initVal = 0;

  for (int i = 0; i < 512; ++i) host_norm_integrals[i] = 1.0;
  cudaMemcpyToSymbol(dev_norm_integrals, host_norm_integrals,
                     512*sizeof(double));

  for (unsigned int i = 0; i < fcns_to_normalise.size(); ++i) {
    unsigned int fcnIdx = fcns_to_normalise[i].first;
    unsigned int parIdx = fcns_to_normalise[i].second;
    GeneralFcn normaliser(fcnIdx, parIdx, false);
    initVal = 0;
    double integral = thrust::transform_reduce(dev_norm_data->begin(),
                                               dev_norm_data->end(),
                                               normaliser,
                                               initVal,
                                               thrust::plus<double>());
    integral *= 0.001; // (Cheating by hardcoding step size!)
    host_norm_integrals[parIdx] = 1.0 / integral;
  }
  cudaMemcpyToSymbol(dev_norm_integrals, host_norm_integrals,
                     512*sizeof(double));
  // PDF evaluation follows
}
```

```
// In main:
// Create normalisation-integral points
host_data.clear();
for (double xval = -5.0; xval < 5.0; xval += 0.001) {
  host_data.push_back(xval);
}
dev_norm_data = new device_vector<double>(host_data);

// (...)

host_fcnIdx = atoi(argv[1]);
std::pair<unsigned int, unsigned int> fInfo(host_fcnIdx, 0);
fcns_to_normalise.push_back(fInfo);
```

Finally - GooFit!

- **The example and exercise code is getting complex already - it has reached the threshold where object orientation would be useful in maintaining it.**
- **Time to introduce the already-written OO framework!**
- **We have not yet dealt with:**
  - **Dependence on multiple variables**
  - **Non-NLL goodness-of-fit metrics**
  - **Binned data**
  - **Plotting**

Sketch of solutions

- **Multiple variables: Instead of passing `xval`, pass a pointer to an array of event data. Then let the `dev_indices` array store, in addition to the parameters, the indices of the variables on which the function depends.**
- **Other metrics: Add an additional function pointer, so we have one function to calculate the PDF and one to take the metric.**
- **Binned data: Make a separate `operator` function which takes data in the format (bin center, bin content).**
- **Plotting: Provide a function to evaluate the PDF at specified points and return an array of the values.**

Simple Gaussian fit

- **From `example4b.cu` (this and subsequent examples are in the `release_16Jan2013` subdirectory):**

```
int main (int argc, char** argv) {
  Variable* xvar = new Variable("xvar", -5, 5);

  TRandom donram(42);
  UnbinnedDataSet data(xvar);
  for (int i = 0; i < 10000; ++i) {
    fptype val = donram.Gaus(0.2, 1.1);
    if (fabs(val) > 5) {--i; continue;}
    data.addEvent(val);
  }

  Variable* mean = new Variable("mean", 0, 1, -10, 10);
  Variable* sigm = new Variable("sigm", 1, 0.5, 1.5);
  GaussianThrustFunctor gauss("gauss", xvar, mean, sigm);

  gauss.setData(&data);
  PdfFunctor fitter(&gauss);
  fitter.fit();

  return 0;
}
```

Data sets

- **`UnbinnedDataSet` and `BinnedDataSet` both constructed from a container of `Variable` pointers:**

```
vector<Variable*> myVars;
Variable* massD0 = new Variable("massD0", -5, 5);
Variable* deltam = new Variable("deltam", -5, 5);
myVars.push_back(massD0);
myVars.push_back(deltam);

UnbinnedDataSet myData(myVars);
BinnedDataSet myBinnedData(myVars);
```

- **Fill with `addEvent` method:**

```
for (int i = 0; i < numData; ++i) {
  massD0->value = getMassD0(i);
  deltam->value = getDeltaM(i);
  myData->addEvent();
}
```

Different metrics

- **Goodness-of-fit metric is encapsulated in the `FitControl` subclasses:**

```
UnbinnedNllFit
BinnedNllFit   // Poisson prob of observed events in bin
BinnedErrorFit // User-provided errors
BinnedChisqFit // Error = sqrt(n)
```

- **Default is to use `UnbinnedNllFit` if provided with a `UnbinnedDataSet`, and `BinnedNllFit` if given `BinnedDataSet`.**

- **Example of device code:**

```
__device__ fptype calculateBinWithError (fptype rawPdf,
                                         fptype* evtVal,
                                         unsigned int par) {
  // In this case interpret the rawPdf as just a number,
  // not a number of events.
  // evtVal should have the structure (bin entry, bin error).

  rawPdf -= evtVal[0]; // Subtract observed value.
  rawPdf /= evtVal[1]; // Divide by error.
  rawPdf *= rawPdf;
  return rawPdf;
}
```

Example of chisquare fit

- **Code from `example4c.cu`, which makes toy MC of $D^0 \to K\pi$ decays and then fits the ratio of right-sign to wrong-sign events.**
- **Compile thus:**

```
qsub -I -l nodes=1:ppn=12:gpus=2 -l walltime=01:00:00
cd $TMPDIR
module load cuda
export ROOTSYS=/nfs/10/ucn1122/root_53006/
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:./rootstuff/
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${ROOTSYS}/lib/
cp -r /nfs/10/ucn1122/goofitcourse/release_16Jan2013/* .
gmake example4c
./example4c
```

Example 4c code

- **Creating binned dataset with errors:**

```
BinnedDataSet* ratioData = new BinnedDataSet(decayTime);
for (unsigned int i = 0; i < wsEvts.size(); ++i) {
  double ratio = wsEvts[i];
  if (0 == rsEvts[i]) rsEvts[i] = 1; // Cheating to avoid div by zero.
  ratio /= rsEvts[i];

  if (0 == wsEvts[i]) wsEvts[i] = 1; // Avoid zero errors
  double error = wsEvts[i] / pow(rsEvts[i], 2);
  error       += pow(wsEvts[i], 2) / pow(rsEvts[i], 3);
  error        = sqrt(error);

  ratioData->setBinContent(i, ratio);
  ratioData->setBinError(i, error);
}
```

- **Setting the fit metric:**

```
PolynomialThrustFunctor* poly =
  new PolynomialThrustFunctor("poly", decayTime, weights);
poly->setFitControl(new BinnedErrorFit());
poly->setData(ratioData);
PdfFunctor* datapdf = new PdfFunctor(poly);
datapdf->fit();
```

Let's make some plots!

- **Use ROOT for actual plotting, but we still need to extract the numbers.**
- **Still from `example4c.cu`:**

```
vector<fptype> values;
poly->evaluateAtPoints(decayTime, values);
TH1D pdfHist("pdfHist", "", decayTime->numbins,
                            decayTime->lowerlimit,
                            decayTime->upperlimit);
for (int i = 0; i < values.size(); ++i) {
  pdfHist.SetBinContent(i+1, values[i]);
}
```

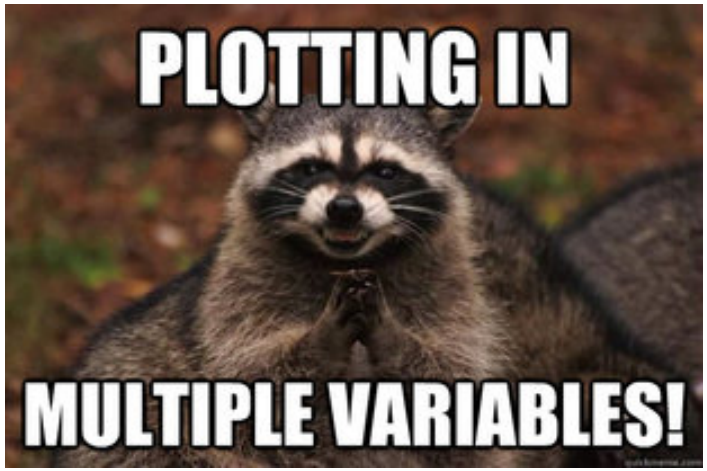
- **`evaluateAtPoints` method just takes the PDF at the center of every bin of the `Variable` we give it (code from `ThrustPdfFunctor.cu`):**

```
UnbinnedDataSet tempdata(observables);
double step = (var->upperlimit - var->lowerlimit) / var->numbins;
for (int i = 0; i < var->numbins; ++i) {
  var->value = var->lowerlimit + (i+0.5)*step;
  tempdata.addEvent();
}
setData(&tempdata);
```

- **Notice that this returns raw (unnormalised) function value, not a PDF.**

Components

- `evaluateAtPoints` **is ok for one dimension, but does not project correctly in several. Use** `getCompProbsAtDataPoints` **instead.**
- **Returns PDF values (normalised) of individual components; from** `example4d.cu` **(also an example of a 2D fit):**




```
// Create 2D PDF
Variable* xmean = new Variable("xmean", 0, 1, -10, 10);
Variable* xsigm = new Variable("xsigm", 1, 0.5, 1.5);
GaussianThrustFunctor xgauss("xgauss", xvar, xmean, xsigm);

Variable* ymean = new Variable("ymean", 0, 1, -10, 10);
Variable* ysigm = new Variable("ysigm", 1, 0.5, 1.5);
GaussianThrustFunctor ygauss("ygauss", yvar, ymean, ysigm);

vector<FunctorBase*> comps;
comps.push_back(&xgauss);
comps.push_back(&ygauss);
ProdThrustFunctor total("total", comps);
```

Evaluating on a grid

- **Create a fake dataset of grid points:**

```
UnbinnedDataSet grid(vars);
for (int i = 0; i < xvar->numbins; ++i) {
  double step = (xvar->upperlimit - xvar->lowerlimit);
  step /= xvar->numbins;
  xvar->value = xvar->lowerlimit + (i + 0.5) * step;
  for (int j = 0; j < yvar->numbins; ++j) {
    step = (yvar->upperlimit - yvar->lowerlimit);
    step /= yvar->lowerlimit + yvar->numbins;
    yvar->value = (j + 0.5) * step;
    grid.addEvent();
  }
}
```

- **Extract PDF values:**

```
fitter.getMinuitValues();
total.setData(&grid);
vector<vector<double> > pdfVals;
total.getCompProbsAtDataPoints(pdfVals);
```

- **Notice the vector-of-vectors structure - zeroth vector is total PDF, subsequent ones are components.**

From vectors to plots

- **Extract PDF values, put in histograms:**

```
for (int i = 0; i < grid.getNumEvents(); ++i) {
  grid.loadEvent(i);
  pdfHist.Fill(xvar->value, yvar->value, pdfVals[0][i]);
  xvarHist.Fill(xvar->value, pdfVals[0][i]);
  yvarHist.Fill(yvar->value, pdfVals[0][i]);
}
```

- **Notice: In this case of a `ProdThrustFunctor`, the component values are not useful!**
- **More useful in `example4e.cu`, showing the same thing for a sum:**

```
for (int i = 0; i < grid.getNumEvents(); ++i) {
  grid.loadEvent(i);
  pdfHist.Fill(xvar->value, pdfVals[0][i]);
  sigHist.Fill(xvar->value, pdfVals[1][i]);
  bkgHist.Fill(xvar->value, pdfVals[2][i]);
  totalPdf += pdfVals[0][i];
}
```

- **Components are normalised so their integral is one - to get signal and background weights, multiply by appropriate `Variable` value.**