```cpp
#include <numeric>
#include "Session2/Session2.hpp"

using namespace cfl;
using namespace std;

namespace NDiscountYieldInterp
{
  std::function<double(double, double)>
  yield(double dInitialTime)
  {
    return [dInitialTime](double dT, double dD) {
      PRECONDITION(dT > dInitialTime + cfl::EPS);
      return -std::log(dD) / (dT - dInitialTime);
    };
  }

} // namespace NDiscountYieldInterp

using namespace NDiscountYieldInterp;

cfl::Function prb::
    discountYieldInterp(const std::vector<double> &rTimes,
                        const std::vector<double> &rDiscount, double dR,
                        double dInitialTime, const cfl::Interp &rInterp)
{
  PRECONDITION(rTimes.size() == rDiscount.size());
  PRECONDITION(rTimes.size() > 0);
  PRECONDITION(rTimes.front() > dInitialTime);
  PRECONDITION(std::is_sorted(rTimes.begin(), rTimes.end(),
                              std::less_equal<double>()));

  //times for interpolation = initial time + discount times
  std::vector<double> uTimes(rTimes.size() + 1);
  uTimes.front() = dInitialTime;
  std::copy(rTimes.begin(), rTimes.end(), uTimes.begin() + 1);

  //yields for interpolation
  std::vector<double> uYields(uTimes.size());
  uYields.front() = dR;
  std::transform(rTimes.begin(), rTimes.end(), rDiscount.begin(),
                 uYields.begin() + 1, yield(dInitialTime));

  //interpolation of market yields
  Function uYieldFunction =
      rInterp.interpolate(uTimes.begin(), uTimes.end(), uYields.begin());

  return cfl::Data::discount(uYieldFunction, dInitialTime);
}
```

```cpp
#include "Session2/Session2.hpp"
#include "cfl/Interp.hpp"
#include "cfl/Macros.hpp"
#include <cmath>

using namespace std;
using namespace cfl;

cfl::Function prb::
forwardLogLinInterp(double dSpot,
                    const std::vector<double> &rTimes,
                    const std::vector<double> &rForwards,
                    double dInitialTime)
{
  PRECONDITION(rTimes.size() == rForwards.size());
  PRECONDITION(rTimes.size() > 0);
  PRECONDITION(rTimes.front() > dInitialTime);
  PRECONDITION(std::is_sorted(rTimes.begin(), rTimes.end(),
                              std::less_equal<double>()));

  //times for interpolation = initial time + delivery times
  std::vector<double> uTimes(rTimes.size()+1);
  uTimes.front() = dInitialTime;
  copy(rTimes.begin(), rTimes.end(), uTimes.begin()+1);

  //logarithm of forward prices
  std::vector<double> uLogForward(uTimes.size());
  uLogForward.front() = std::log(dSpot);
  std::transform(rForwards.begin(), rForwards.end(),
                 uLogForward.begin() + 1,
                 [](double dX) { return std::log(dX); });

  //linear interpolation of the logarithm of forward prices
  cfl::Interp uLin = NInterp::linear();
  Function uLogForwardFunction =
    uLin.interpolate(uTimes.begin(), uTimes.end(), uLogForward.begin());

  return cfl::exp(uLogForwardFunction);
}
```

```cpp
#include "Session2/Session2.hpp"
#include <numeric>

using namespace cfl;
using namespace std;

namespace NForwCarryFit
{
  std::function<double(double, double)>
  costOfCarry(double dSpot, double dInitialTime)
  {
    return [dSpot, dInitialTime](double dF, double dT) {
      PRECONDITION(dT > dInitialTime + cfl::EPS);
      return std::log(dF / dSpot) / (dT - dInitialTime);
    };
  }

} // namespace NForwCarryFit

using namespace NForwCarryFit;

cfl::Function prb::
    forwardCarryFit(double dSpot,
                    const std::vector<double> &rTimes,
                    const std::vector<double> &rForwards,
                    double dInitialTime, cfl::Fit &rFit, cfl::Function &rErr)
{
  PRECONDITION(rTimes.size() == rForwards.size());
  PRECONDITION(rTimes.size() > 0);
  PRECONDITION(rTimes.front() > dInitialTime + cfl::EPS);
  PRECONDITION(std::is_sorted(rTimes.begin(), rTimes.end(),
                              std::less_equal<double>()));

  //market cost-of-carry rates
  std::vector<double> uCostOfCarry(rTimes.size());
  std::transform(rForwards.begin(), rForwards.end(), rTimes.begin(),
                 uCostOfCarry.begin(), costOfCarry(dSpot, dInitialTime));

  rFit.assign(rTimes.begin(), rTimes.end(), uCostOfCarry.begin());
  Function uT([dInitialTime](double dT) { return dT - dInitialTime; }, dInitialTime);
  Function uForward = dSpot * exp(rFit.fit() * uT);
  rErr = uForward * uT * rFit.err();

  return uForward;
}
```

```cpp
#include <numeric>
#include "Session2/Session2.hpp"

using namespace cfl;
using namespace std;

namespace NForwBlackFit
{
  cfl::Function yieldShape1(double dLambda, double dInitialTime)
  {
    std::function<double(double)> uShape =
        [dLambda, dInitialTime](double dT) {
          double dX = std::max(dLambda * (dT - dInitialTime), cfl::EPS);
          return (1 - std::exp(-dX)) / dX;
        };
    return Function(uShape, dInitialTime);
  }
} // namespace NForwBlackFit

using namespace NForwBlackFit;

cfl::Function prb::
    forwardBlackFit(double dSpot, const std::vector<double> &rTimes,
                    const std::vector<double> &rForwards,
                    double dLambda, double dSigma, double dInitialTime,
                    cfl::Function &rErr, cfl::FitParam &rParam)
{
  PRECONDITION((dLambda > 0) && (dSigma >= 0));
  PRECONDITION(rTimes.front() > dInitialTime);

  //construct basis functions and fitting engine
  Function uBase = yieldShape1(dLambda, dInitialTime);
  cfl::Fit uFit = cfl::NFit::linear(uBase);

  //construct the data for fit
  std::vector<double> uY(rTimes.size());
  Function uF = yieldShape1(2 * dLambda, dInitialTime);
  uF *= (0.5 * dSigma * dSigma);
  std::transform(rTimes.begin(), rTimes.end(), rForwards.begin(), uY.begin(),
                 [uF, dSpot, dInitialTime](double dT, double dF) {
                   double dY = std::log(dF / dSpot) / (dT - dInitialTime) + uF(dT);
                   return dY;
                 });
  uFit.assign(rTimes.begin(), rTimes.end(), uY.begin());
  rParam = uFit.param();
  Function uT([dInitialTime](double dT) { return dT - dInitialTime; }, dInitialTime);
  Function uForward = dSpot * exp((uFit.fit() - uF) * uT);
  rErr = uForward * uT * uFit.err();
  return uForward;
}
```