

01/31/2019 • 16 minutes to read

## In this article

[An Example](#)

[The Four Key Prediction Market Equations](#)

[The Demo Program](#)

[Wrapping Up](#)

---

June 2016

Volume 31 Number 6

[Test Run]

# Introduction to Prediction Markets

By [James McCaffrey](#)



Suppose you want to predict the outcome of an upcoming championship football game between the Xrays and the Yanks. You find a group of 20 football experts and give each of them \$500 in tokens. The experts are allowed to buy and sell shares of each of the two teams, in a way that's somewhat similar to how the stock market works.

When an expert buys shares in one team, say the Xrays, the price of a share of that team increases and the price of a share of the other team decreases. Over time, the experts will buy and sell shares of the two teams until prices stabilize, and then you'll be able to infer the probability of each team winning.

You halt trading the day before the championship game. After the game is played and the winner is determined, you pay experts who have shares in the winning team according to the last price of the team when trading closed. Because the experts know they'll be paid, they have incentive to give their true opinions during trading.


What I've just described is called a prediction market. In this article, I'll describe the math behind prediction markets and show you how to implement the key functions in code. It's unlikely you'll ever have to create a prediction market in your day-to-day job, but I think you'll find the ideas very interesting. Additionally, some of the programming techniques presented in this article can be used in more common software development scenarios.

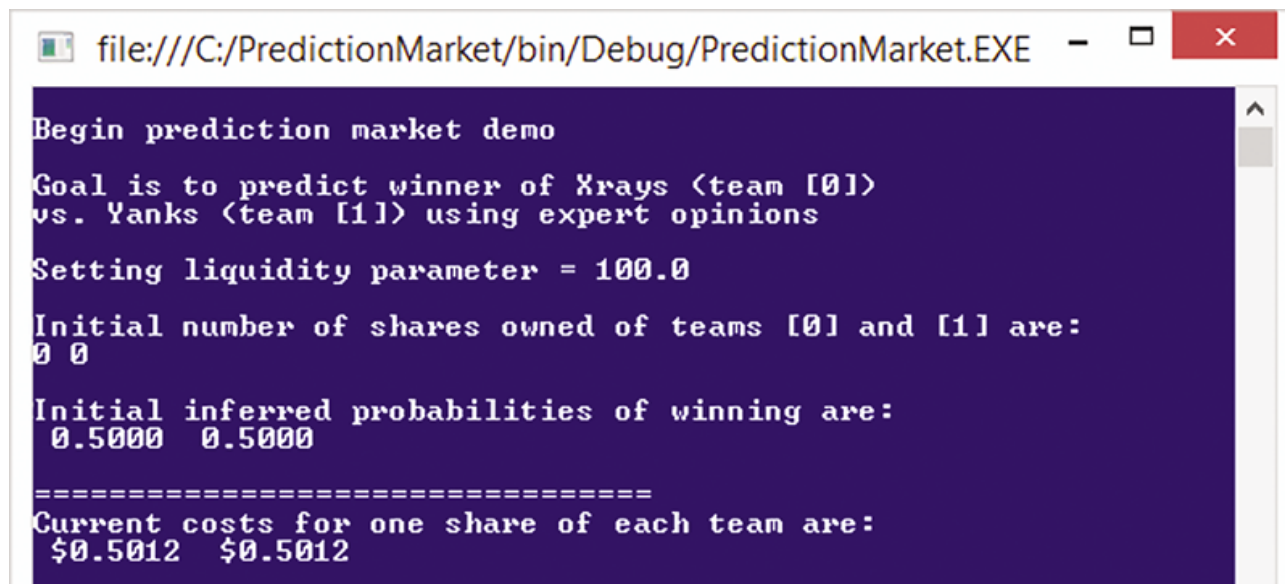
This article assumes you have at least beginner-level coding skill, but doesn't assume you know anything about prediction markets. I present a complete demo program, and you can also get the source code from the download that accompanies this article. The demo uses C#, but you should have no trouble refactoring the code to another language if you wish.

Note that this is an informal introduction to prediction markets, intended primarily for software developers. I take some liberties with terminology and definitions in order to keep the main ideas as clear as possible.

## An Example

Prediction markets are perhaps best explained with a concrete example. Take a look at the demo program in **Figure 1**. After some preliminary messages, the demo output starts with:

XML	 Copy
<pre>Setting liquidity parameter = 100.0  Initial number of shares owned of teams [0] and [1] are: 0 0  Initial inferred probabilities of winning are: 0.5000 0.5000</pre>	



```
Update: expert [01] buys 20 shares of team [0]
Cost of transaction to expert was: $10.50

New number of shares owned of teams [0] and [1] are:
20 0

New inferred probabilities of winning are:
0.5498 0.4502

=====
Current costs for one share of each team are:
$0.5511 $0.4514

Update: expert [02] buys 20 shares of team [1]
Cost of transaction to expert was: $9.50

New number of shares owned of teams [0] and [1] are:
20 20

New inferred probabilities of winning are:
0.5000 0.5000

=====
Current costs for one share of each team are:
$0.5012 $0.5012

Update: expert [03] buys 60 shares of team [0]
Cost of transaction to expert was: $34.43

New number of shares owned of teams [0] and [1] are:
80 20

New inferred probabilities of winning are:
0.6457 0.3543

=====
Current costs for one share of each team are:
$0.6468 $0.3555

Update: expert [01] sells 10 shares of team [0]
Cost of transaction to expert was: $-6.34

New number of shares owned of teams [0] and [1] are:
70 20

New inferred probabilities of winning are:
0.6225 0.3775

=====
Update: Market Closed


End prediction market demo
```

**Figure 1 A Prediction Market Demo**

The liquidity parameter will be explained in detail shortly, but for now it's enough to know that liquidity controls how much market prices react to buying and selling. Larger values of liquidity produce smaller changes in prices.


Initially, no shares are owned by the experts. Because the number of shares owned for each team is the same (zero), it's reasonable that the initial inferred probability a team will win is 0.50.

The next part of the demo output is:


XML	 Copy
<pre>Current costs for one share of each team are: \$0.5012 \$0.5012</pre>	

At any point in time, a share of each team has a certain price. Experts need to know this price because they're playing for real money. Because the initial probabilities of winning are equal, it's reasonable that the prices for a share of each team are also the same.

The next part of the demo output is:


XML	 Copy
<pre>Update: expert [01] buys 20 shares of team [0]  Cost of transaction to expert was: \$10.50</pre>	

Expert #1 believes that team 0, the Xrays, will win and buys 20 shares of team 0. The cost to the expert is \$10.50. Notice that the price for 20 shares (\$10.50) is not the same as 20 times the price of a single share ( $20 * \$0.5012 = \$10.02$ ). As each share is purchased, the price for an additional share of the team increases. The next part of the demo output is:


XML	 Copy
<pre>New number of shares owned of teams [0] and [1] are: 20 0  New inferred probabilities of winning are: 0.5498 0.4502</pre>	

The demo displays the updated number of shares outstanding on each team,  $(x, y) = (20, 0)$  and computes and displays updated inferred probabilities of each team winning (0.55, 0.45). Because experts have bought more shares of team 0 than team 1, the inferred probability of team 0 winning must be greater than that of team 1. The calculation of the probabilities will be explained shortly.

Next, the demo displays:


XML	 Copy
<p>Current costs for one share of each team are: \$0.5511 \$0.4514</p> <p>Update: expert [02] buys 20 shares of team [1]</p> <p>Cost of transaction to expert was: \$9.50</p>	

The new cost per share for each team is calculated and displayed. Notice that the price of a share of team 0 (\$0.55) is now quite a bit more expensive than that of team 1 (\$0.45). This gives experts an incentive to buy shares of team 1 if they think the price is a good value relative to the likelihood of team 1 winning. In this case, the demo simulates expert #2 buying 20 shares of team 1 for a cost of \$9.50. Next:

XML	 Copy
<p>New number of shares owned of teams [0] and [1] are: 20 20</p> <p>New inferred probabilities of winning are: 0.5000 0.5000</p>	

There are now 20 shares outstanding for each team, so the inferred probabilities of each team winning revert to 0.50 and 0.50.


The next part of the demo output is:

XML	 Copy
<p>Current costs for one share of each team are: \$0.5012 \$0.5012</p> <p>Update: expert [03] buys 60 shares of team [0] Cost of transaction to expert was: \$34.43</p> <p>New number of shares owned of teams [0] and [1] are: 80 20</p> <p>New inferred probabilities of winning are: 0.6457 0.3543</p>	

Expert #3 believes strongly that team 0 will win, so he buys 60 shares of team 0 for a cost of \$34.43. This transaction changes the number of outstanding shares to (80, 20) and

causes the new inferred probabilities of winning to move strongly toward team 0 (0.65, 0.35).

Next, expert #1 sees that the value of his shares in team 0 have risen greatly to approximately \$0.6468 per share:

XML	 Copy
<pre>Current costs for one share of each team are: \$0.6468 \$0.3555  Update: expert [01] sells 10 shares of team [0] Cost of transaction to expert was: \$-6.34  New number of shares owned of teams [0] and [1] are: 70 20  New inferred probabilities of winning are: 0.6225 0.3775</pre>	

Expert #1 feels that team 0 is now somewhat overpriced relative to its chances of winning and sells 10 of his 20 shares, getting \$6.34 (indicated by the negative sign). The new inferred probabilities adjust back to a bit more equal, but team 0 is still predicted to win with probability 0.63.

The demo ends by closing trading. The final probabilities are the goal of the prediction market. After the game between the Xrays and the Yanks is played, experts would be paid for shares they hold in the winning team, based on the final share price of the winning team. The payments encourage the experts to give their true opinions.

## The Four Key Prediction Market Equations

A basic prediction market uses four math equations, as shown in **Figure 2**. Bear with me; the equations aren't nearly as complicated as they might first appear. There are several math models that can be used to define a prediction market. The model presented in this article is based on what's called the Logarithmic Market Scoring Rule (LMSR).

$$\begin{aligned}
 (1) \quad C(x, y) &= b * \ln(e^{x/b} + e^{y/b}) \\
 (2) \quad C_{trans} &= C_{after} - C_{before} \\
 (3) \quad p_x(x, y) &= \frac{e^{x/b}}{e^{x/b} + e^{y/b}} \\
 (4) \quad p_y(x, y) &= \frac{e^{y/b}}{e^{x/b} + e^{y/b}}
 \end{aligned}$$


**Figure 2 The Four Key Prediction Market Equations**

Equation 1 is the cost function associated with a set of outstanding shares (x, y). The equation, which isn't at all obvious, comes from economics theory. From a developer's point of view, you can think of the equation as a helper function. It accepts x, which is the number of shares held of option 0, and y, which is the number of shares held of option 1, and returns a value. Variable b in all four equations is the liquidity parameter. Suppose x = 20 and y = 10. If b = 100.0, then  $C(x, y) = 100.0 * \ln(\exp(20/100) + \exp(10/100)) = 100.0 * \ln(1.22 + 1.11) = 100.0 * 0.8444 = \$84.44$ . The return value is used in equation 2.

Equation 2 is the cost of a transaction to a buyer. Suppose a current set of outstanding shares is (20, 10) and an expert buys 30 shares of option 0. The cost of that transaction to the expert is computed using equation 2 as  $C(20+30, 10) - C(20, 10) = C(50, 10) - C(20, 10) = 101.30 - 84.44 = \$16.86$ . If an expert sells shares, the cost of the transaction will be a negative value indicating the expert is paid.

Equation 3 is technically the marginal price of option 0 based on a set of outstanding shares (x, y). But a marginal price can be loosely interpreted as the probability that an option will win. Equation 4 is the marginal price (probability) of option 1. If you look at the two equations closely, you'll notice they must sum to 1.0, as is required for a set of probabilities.


Implementing the four key prediction market equations is straightforward. The demo program implements the cost, equation 1, as:

C#	 Copy
<pre>static double Cost(int[] outstanding, double liq) {     double sum = 0.0;     for (int i = 0; i &lt; 2; ++i)         sum += Math.Exp(outstanding[i] / liq);     return liq * Math.Log(sum); }</pre>	


```
sum += Math.Exp(outstanding[1] / liq);  
return liq * Math.Log(sum);  
}
```

The Cost method is virtually an exact translation of equation 1. Notice method Cost assumes there are just two options. For simplicity, no error checking is performed.

Equation 2 is also rather simple to implement:


C#	 Copy
<pre>static double CostOfTrans(int[] outstanding, int idx, int nShares, double liq) {     int[] after = new int[2];     Array.Copy(outstanding, after, 2);     after[idx] += nShares;     return Cost(after, liq) - Cost(outstanding, liq); }</pre>	

The array named after holds the new number of outstanding shares after a transaction, and the method then just calls the Cost helper method twice. With a method to calculate the cost of a transaction in hand, it's easy to write a method that calculates the cost of buying a single share of each of the two options:

C#	 Copy
<pre>static double[] CostForOneShare(int[] outstanding, double liq) {     double[] result = new double[2];     result[0] = CostOfTrans(outstanding, 0, 1, liq);     result[1] = CostOfTrans(outstanding, 1, 1, liq);     return result; }</pre>	

The cost of a single share can be used by experts to get an approximation of how much it would cost to buy n shares of an option.

Method Probabilities returns the two marginal prices (inferred probabilities) of each option winning in an array:

C#	 Copy
<pre>static double[] Probabilities(int[] outstanding, double liq) {     double[] result = new double[2];     result[0] = CostForOneShare(outstanding, liq)[0];     result[1] = CostForOneShare(outstanding, liq)[1];     return result; }</pre>	



```

double denom = 0.0;
for (int i = 0; i < 2; ++i)
    denom += Math.Exp(outstanding[i] / liq);
for (int i = 0; i < 2; ++i)
    result[i] = Math.Exp(outstanding[i] / liq) / denom;
return result;
}

```

If you compare the code for method Probabilities with equations 3 and 4, you'll see that, again, the code follows directly from the math definition.


## The Demo Program

To create the demo program, I launched Visual Studio and selected the C# console application program template. I named the project PredictionMarket. The demo has no significant Microsoft .NET Framework dependencies, so any version of Visual Studio will work.

After the template code loaded, in the Solution Explorer window I renamed file Program.cs to the more descriptive PredictionMarketProgram.cs and allowed Visual Studio to automatically rename class Program for me. At the top of the source code, I deleted all using statements that referenced unneeded .NET namespaces, leaving just the reference to the top-level System namespace.

The complete demo code, with a few minor edits and some WriteLine statements deleted to save space, is presented in **Figure 3**. All the program control logic is in the Main method. All the prediction market functionality is in four static methods, and there are two ShowVector helper display methods.

Figure 3 Prediction Market Demo

C#	 Copy
<pre> using System; namespace PredictionMarket {     class PredictionMarketProgram     {         static void Main(string[] args)         {             Console.WriteLine("Begin prediction market demo ");             Console.WriteLine("Goal is to predict winner of Xrays");             Console.WriteLine("vs. Yanks using expert opinions");             double liq = 100.0; </pre>	

```

Console.WriteLine("Setting liquidity parameter = " +
    liq.ToString("F1"));
int[] outstanding = new int[] { 0, 0 };
Console.WriteLine("Initial number of shares owned are:");
ShowVector(outstanding);
double[] probs = Probabilities(outstanding, liq);
Console.WriteLine("Initial probabilities of winning:");
ShowVector(probs, 4, " ");
Console.WriteLine("=====");
double[] costPerShare = CostForOneShare(outstanding, liq);
Console.WriteLine("Current costs for one share are: ");
ShowVector(costPerShare, 4, " $");
Console.WriteLine("Update: expert [01] buys 20 shares " +
    "of team [0]");
double costTrans = CostOfTrans(outstanding, 0, 20, liq);
Console.WriteLine("Cost of transaction to expert was: $" +
    costTrans.ToString("F2"));
outstanding = new int[] { 20, 0 };
Console.WriteLine("New number of shares owned are: ");
ShowVector(outstanding);
probs = Probabilities(outstanding, liq);
Console.WriteLine("New inferred probs of winning:");
ShowVector(probs, 4, " ");
Console.WriteLine("=====");
costPerShare = CostForOneShare(outstanding, liq);
Console.WriteLine("Current costs for one share are:");
ShowVector(costPerShare, 4, " $");
Console.WriteLine("Update: expert [02] buys 20 shares " +
    "of team [1]");
costTrans = CostOfTrans(outstanding, 1, 20, liq);
Console.WriteLine("Cost of transaction to expert was: $" +
    costTrans.ToString("F2"));
outstanding = new int[] { 20, 20 };
Console.WriteLine("New number of shares owned are:");
ShowVector(outstanding);
probs = Probabilities(outstanding, liq);
Console.WriteLine("New inferred probs of winning:");
ShowVector(probs, 4, " ");
Console.WriteLine("=====");
costPerShare = CostForOneShare(outstanding, liq);
Console.WriteLine("Current costs for one share are:");
ShowVector(costPerShare, 4, " $");
Console.WriteLine("Update: expert [03] buys 60 shares " +
    "of team [0]");
costTrans = CostOfTrans(outstanding, 0, 60, liq);
Console.WriteLine("Cost of transaction to expert was: $" +
    costTrans.ToString("F2"));
outstanding = new int[] { 80, 20 };
Console.WriteLine("New number of shares owned are:");
ShowVector(outstanding);
probs = Probabilities(outstanding, liq);

```

```

Console.WriteLine("New inferred probs of winning:");
ShowVector(probs, 4, " ");
Console.WriteLine("=====");
costPerShare = CostForOneShare(outstanding, liq);
Console.WriteLine("Current costs for one share are: ");
ShowVector(costPerShare, 4, " $");
Console.WriteLine("Update: expert [01] sells 10 shares " +
    "of team [0]");
costTrans = CostOfTrans(outstanding, 0, -10, liq);
Console.WriteLine("Cost of transaction to expert was: $" +
    costTrans.ToString("F2"));
outstanding = new int[] { 70, 20 };
Console.WriteLine("New number of shares owned are:");
ShowVector(outstanding);
probs = Probabilities(outstanding, liq);
Console.WriteLine("New inferred probs of winning:");
ShowVector(probs, 4, " ");
Console.WriteLine("=====");
Console.WriteLine("Update: Market Closed");
Console.WriteLine("\nEnd prediction market demo \n");
Console.ReadLine();
} // Main()
static double[] Probabilities(int[] outstanding,
    double liq)
{
    double[] result = new double[2];
    double denom = 0.0;
    for (int i = 0; i < 2; ++i)
        denom += Math.Exp(outstanding[i] / liq);
    for (int i = 0; i < 2; ++i)
        result[i] = Math.Exp(outstanding[i] / liq) / denom;
    return result;
}
static double Cost(int[] outstanding, double liq)
{
    double sum = 0.0;
    for (int i = 0; i < 2; ++i)
        sum += Math.Exp(outstanding[i] / liq);
    return liq * Math.Log(sum);
}
static double CostOfTrans(int[] outstanding, int idx,
    int nShares, double liq)
{
    int[] after = new int[2];
    Array.Copy(outstanding, after, 2);
    after[idx] += nShares;
    return Cost(after, liq) - Cost(outstanding, liq);
}
static double[] CostForOneShare(int[] outstanding,
    double liq)
{

```

```

double[] result = new double[2];
result[0] = CostOfTrans(outstanding, 0, 1, liq);
result[1] = CostOfTrans(outstanding, 1, 1, liq);
return result;
}
static void ShowVector(double[] vector, int dec, string pre)
{
    for (int i = 0; i < vector.Length; ++i)
        Console.Write(pre + vector[i].ToString("F" + dec) + " ");
    Console.WriteLine("\n");
}
static void ShowVector(int[] vector)
{
    for (int i = 0; i < vector.Length; ++i)
        Console.Write(vector[i] + " ");
    Console.WriteLine("\n");
}
} // Program class
} // ns

```

After displaying some preliminary messages, program execution in method Main begins with:

C#

 Copy

```

double liq = 100.0;
int[] outstanding = new int[] { 0, 0 };
ShowVector(outstanding);

```

Variable `liq` is the liquidity parameter. A value of 100.0 is typical, but if you experiment by adjusting the value, you'll see how it affects the change in share prices after a transaction. Larger liquidity values produce smaller changes. The array named `outstanding` holds the total number of shares owned by all experts, on each of the two teams. Notice that the liquidity parameter has to be passed to the four static market prediction methods. An alternative design is to encapsulate the methods into a C# class and define liquidity as a member field.

Next, the number of outstanding shares is used to determine the inferred probabilities of each team winning:

C#


 Copy

```

double[] probs = Probabilities(outstanding, liq);
Console.WriteLine("Initial probabilities of winning:");
ShowVector(probs, 4, " ");


```

Next, the demo displays the costs of buying a single share of each of the two teams:

C#	 Copy
<pre>double[] costPerShare = CostForOneShare(outstanding, liq); Console.WriteLine("Current costs for one share are: "); ShowVector(costPerShare, 4, " \$");</pre>	


In a realistic prediction market, this information would be useful to the market experts to help them assess whether the share price of a team is too high or too low relative to the expert's perception that the team will win.

The demo program simulates one of the experts buying some shares, like so:

C#	 Copy
<pre>Console.WriteLine("Update: expert [01] buys 20 shares of team [0]"); double costTrans = CostOfTrans(outstanding, 0, 20, liq); Console.WriteLine("Cost of transaction to expert was: \$" +     costTrans.ToString("F2"));</pre>	


In a real prediction market, the system would have to maintain quite a bit of information about experts' account balances and the number of shares owned.

Next, the number of outstanding shares is updated, like so:

C#	 Copy
<pre>outstanding = new int[] { 20, 0 }; Console.WriteLine("New number of shares owned on teams [0] " +     "and [1] are: "); ShowVector(outstanding);</pre>	

If you refer back to the math equations in **Figure 2**, you'll notice that the number of outstanding shares for each team/option, (x, y), is needed by all equations.

After the number of outstanding shares has been updated, that information is used to estimate the revised probabilities of each team or option winning:

C#	 Copy
<pre>probs = Probabilities(outstanding, liq); Console.WriteLine("New inferred probabilities of</pre>	

```
    winning are: ");  
    ShowVector(probs, 4, " ");
```

Recall that these values are really marginal prices, but it's useful to think of them as probabilities. Ultimately, the purpose of a prediction market is to produce the likelihood that each team or option will win, so the final set of probabilities after the market stabilizes is what you're after.

The demo program concludes by repeating the following five operations three more times:

- Show current cost for one share of each team
- Perform a buy or sell transaction
- Show the cost of the transaction
- Update the total number of shares outstanding
- Update the probability of each team winning

Notice that the demo program begins with the probabilities of both teams being equal. This isn't realistic in many real prediction-market scenarios. It's possible to initialize a prediction market with unequal probabilities by solving for  $x$  and  $y$  in equations 3 and 4.

## Wrapping Up

The information in this article is based on the 2002 research paper, "Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation," by Robin Hanson. You can find a PDF version of the paper in several places on the Internet by using any search tool.

Prediction markets aren't just an abstract theoretical idea. In the past few years, several companies have been created that actually implement prediction markets for real money.

An area of active research is in what are called combinatorial prediction markets. Instead of picking just one of two options to win, experts can buy shares in combination events such as team A will beat team B and Team J will beat team K. Combinatorial prediction markets are much more complex than simple markets.

---

**Dr. James McCaffrey** works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. Dr. McCaffrey can be reached at [jammc@microsoft.com](mailto:jammc@microsoft.com).

Thanks to the following Microsoft technical experts who reviewed this article: Pallavi Choudhury, Gaz Iqbal, Umesh Madan and Tien Suwandy

---

[Discuss this article in the MSDN Magazine forum](#)