WOLFRAM WHITE PAPER

Identifying Business Cycles Using *RLink* in Wolfram *Finance Platform*™

WOLFRAM

Introduction

RLink is a Wolfram *Finance Platform* application that uses *J/Link*™ and RJava/JRI Java libraries to link to R. It sets up a robust communication channel between Wolfram *Finance Platform* and R, enabling the exchange of data and execution of R code from within Wolfram *Finance Platform*.

We demonstrate the use of *RLink* by identifying business cycles that affect stock markets for periods generally lasting about four years. Although time series functionality is available in Wolfram *Finance Platform*, for the sake of this demonstration we call R's functionality from Wolfram *Finance Platform.*

Motivations for *RLink*

Interoperability with existing and legacy code

While business software systems may be the heart and soul of what differentiates one company's offering from another, the rapid pace of change across the marketplace, new tools and techniques, and customer usage/needs increase the velocity with which code moves to achieve legacy status. Even when newer software is available that is better suited for the task at hand, often it is impractical for businesses to switch to the new system quickly. *RLink* provides robust communication between Wolfram *Finance Platform* and R, giving businesses the opportunity to improve upon their existing investments in R-based infrastructure rather than start over.

Accessing niche add-on libraries

The benefits of linking from Wolfram *Finance Platform* to other languages and tools differ from case to case. For example, creating an Excel link added an alternative interface paradigm to Wolfram *Finance Platform.* But in the case of *RLink* in Wolfram *Finance Platform*, the benefits have very little to do with R, the language. It is not immediately obvious that it does many things that you cannot already do in Wolfram *Finance Platform* or that there are many things it does significantly better. The real benefit is in the connection it makes to the R community.

With *RLink* you have immediate access to the work of the R community through the add-on libraries that they have created to extend R into their fields. A large number of these free libraries fill out many of the niches in finance—sometimes popular, sometimes obscure, but in any case a large number. At a stroke, all of them are made immediately available to the Wolfram *Finance Platform* environment, interpreted through the R language runtime.

Obtaining and Visualizing GNP Data

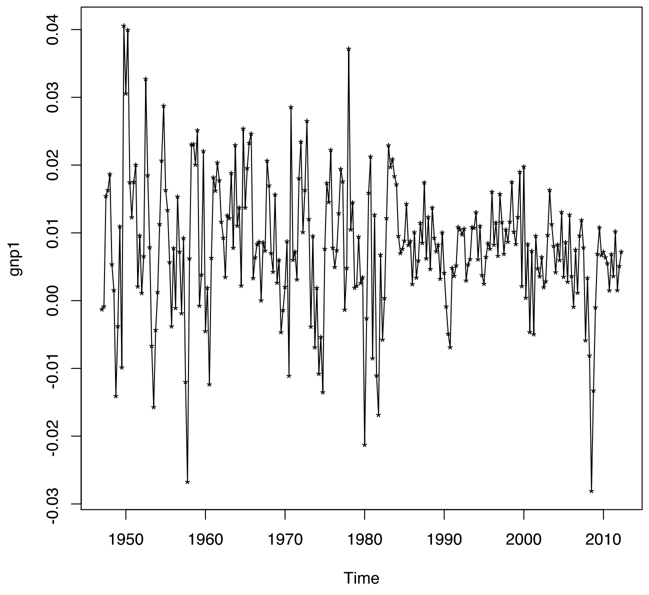
Changes in gross national product (GNP) contribute greatly to business cycles. Economic data such as GNP is time series data. Wolfram *Finance Platform* has built-in time series functions, but in this white paper we will explore this using R. With Wolfram *Finance Platform*'s *RLink*, users can exchange data between *Finance Platform* and R and execute R code from within *Finance Platform*. That means R users can use their existing code and still take advantage of *Finance Platform*'s workflow. To start, we call R from within *Finance Platform*:

```
Needs["RLink`"]
InstallR[]
```

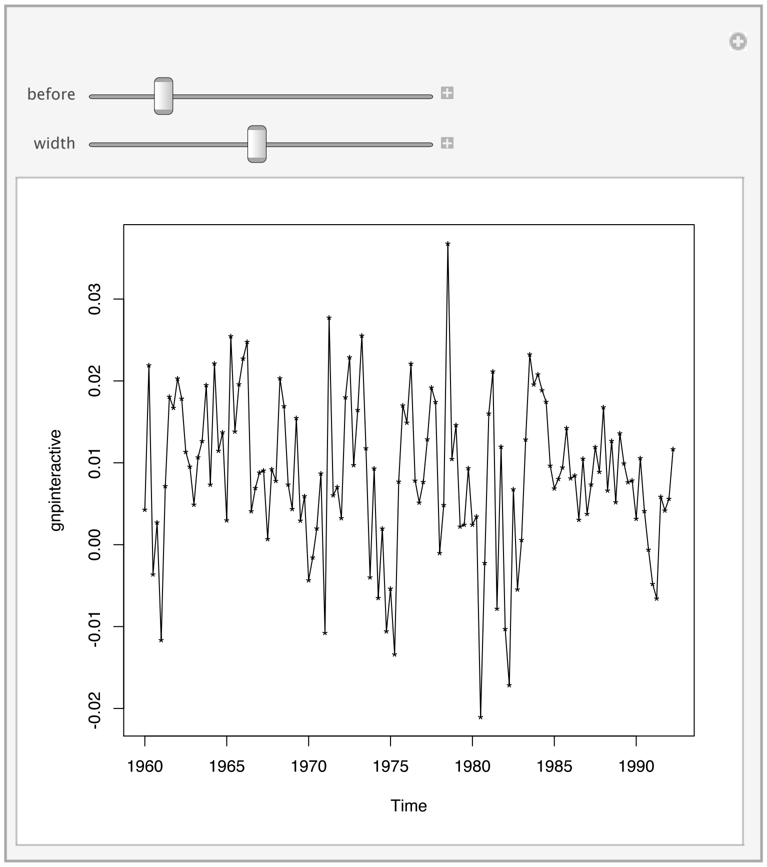
Using Wolfram|Alpha®, we obtain United States GNP data and compute the growth rates[1].

We can visualize this data in *Finance Platform,* but R users who already have a significant code base can send this data to R, using `RSet`, for analysis and visualization[2].

R has built-in analytical plots. Here, we will combine the static plot with *Finance Platform*'s interactivity to create an explorer[3]. Here is the R plot:



It is easy to write a quick command to extend the static plot to an interactive app in *Finance Platform*[4].



Performing Time Series Analysis

One of the common techniques to perform time series analysis is to fit the data with an autoregressive (AR) model. We can do so using *Finance Platform*'s built-in time series commands or we can utilize some of the packages written in R. Here, we will do the analysis in R and verify the result with *Finance Platform*.

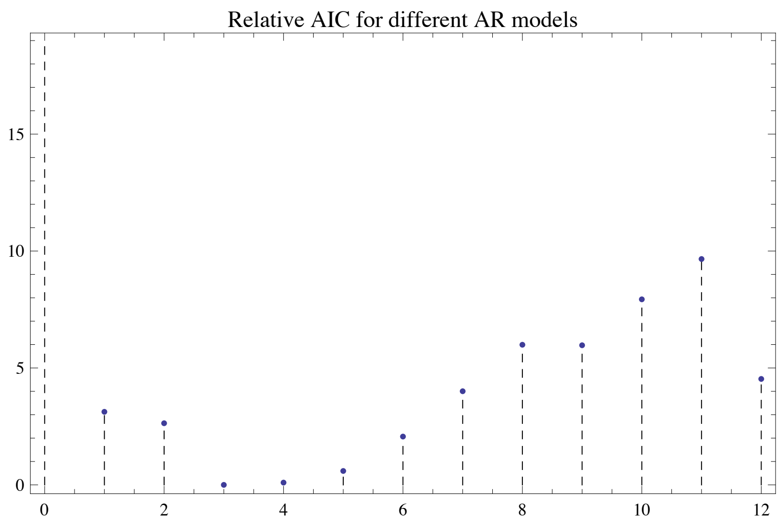
We fit the time series data to an autoregressive (AR) model using the maximum-likelihood estimation in R[5].

What is important in an AR model are its coefficients and order. Combining *Finance Platform*'s string operation with R's analysis, the AR model's coefficients, order, and standard error are determined[6]. From the coefficients, it is clear that the data is fitted using an AR(3) model.

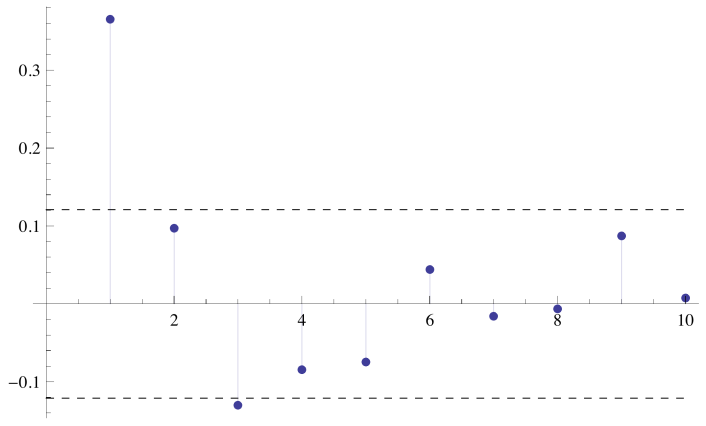
```

Call:
ar(x = gnp, method = "mle")

Coefficients:
      1        2        3
 0.3427   0.1414   −0.1319

Order selected 3  sigma^2 estimated as  8.436e−05
```

The Akaike information criterion (AIC) is used to verify the AR model's order. We can visualize the result in a tabular form[7]. From the graph, it is easy to see that the optimal value of time lag is 3.

Lag	AIC diff
0	38.6016
1	3.12438
2	2.63561
3	0.
4	0.0974919
5	0.594544
6	2.06579
7	4.00438
8	5.99201
9	5.97041
10	7.93484
11	9.65893
12	4.53024



In time series analysis, the partial autocorrelation function (PACF) can be used to verify the extent of the lag in an autoregressive model. We can use *Finance Platform*'s built-in `PartialCorrelationFunction` to carry out this verification[8].



With a 95% confidence interval, we can conclude that the order of AR process is indeed 3.

Once we obtain the order of the AR process, an autoregressive integrated moving-average (ARIMA) model can be applied to find the business cycles[9].

A business cycle turns out to be just about two and half years. From our preliminary analysis, it appears that business cycles are close to the period of cyclical bull and bear stock market cycles. While economists prefer the term "short-run economic fluctuations," to many practitioners, the cycles are made of both short-run and long-run fluctuations, and there are definite opportunities to profit from such a cycle if fluctuations can be identified.

Summary

As our example above illustrated, *RLink* enables easy interoperability with R from within Wolfram *Finance Platform*. It also provides access to several libraries that have R interfaces. By combining *Finance Platform*'s broad range of capabilities with R, *RLink* offers you unique functionality to solve a host of traditional and nontraditional problems in finance. With *RLink*, R users can use thousands of functions from across the full spectrum of Wolfram *Finance Platform*.

Notes

1. Using Wolfram|Alpha, we obtain GNP data and compute the growth rates:

```
gnp = WolframAlpha["Real gnp usa",
    {{"History:GrossNationalProduct:EconomicData", 1},
     "TimeSeriesData"}];
```

```
td = TemporalData[Differences[gnp[[All, 2]]] / Most[gnp[[All, 2]]], {Rest[gnp[[All, 1]]]}]
```

```
TemporalData[ 1 ]
```

2. We can visualize this data in *Finance Platform*. However, for R users who already have a significant code base, we can send this data to R, using `RSet`, for analysis and visualization:

```
data = First@td["States"];

RSet["gnp", data];
```

To test that the “gnp” variable has indeed been assigned, we use `REvaluate`:

```
REvaluate["gnp"] // Short
```

```
{–0.00129184, –0.000843597, ≪258≫, 0.0050972, 0.00721468}
```

Convert the data to a time series object in R:

```
REvaluate["gnp1=ts(gnp,frequency=4,start=c(1947,1))"] // Short
```

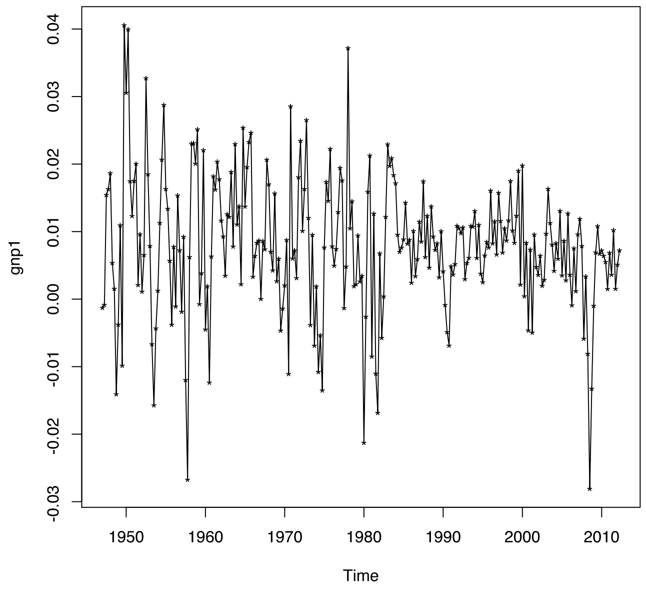
```
RObject({–0.00129184, –0.000843597, ≪259≫, 0.00721468}, RAttributes(≪1≫))
```

3. R has built-in analytical plots. Here, we will combine the static plot with *Finance Platform*’s interactivity to create an explorer. The following functions get R plots into *Finance Platform*:

```
mathematicaRPlotWrapper =
  RFunction["function(filename, plotfun){
pdf(filename)
plotfun()
dev.off()
}"];

Clear[getRPlot];
getRPlot[plotFun_RFunction] :=
  With[{tempfile =
     FileNameJoin[{$TemporaryDirectory, "temp.pdf"}]},
   If[FileExistsQ[tempfile], Quiet@DeleteFile[tempfile]];
   mathematicaRPlotWrapper[tempfile, plotFun];
   If[! FileExistsQ[tempfile], Return[$Failed]];
   Import[tempfile]
  ];
```

This function “plot” takes a sequence of R code statements (a string of R code), and returns the resulting R plot. It works via an intermediate file and reads the resulting image into *Finance Platform*:

```
ClearAll[plot];
plot[code_String] :=
 Show[#, ImageSize → {400, 400}] &@
  getRPlot[
   RFunction["function(){" <> code <> "}"]]

plot["
plot(gnp1)
points(gnp1,pch='*')"
]
```

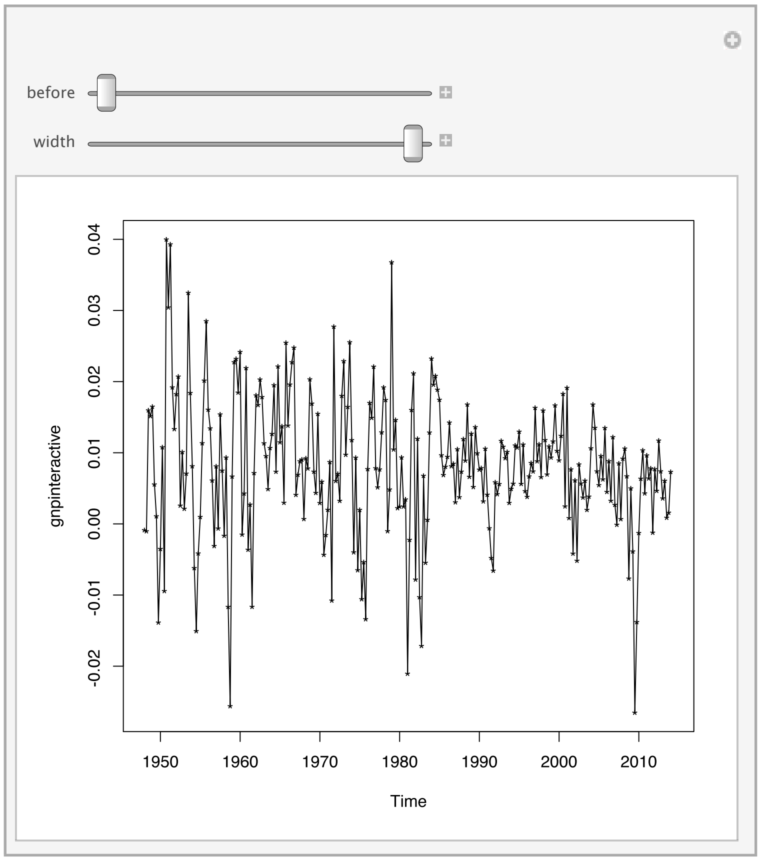


4. It is easy to write a quick command to extend the static plot to an interactive app in *Finance Platform*:

```
ClearAll[plotGnpZoomed]
plotGnpZoomed[before_, width_] :=
 Module[{},
  RSet["before", before];
  RSet["width", width];
  REvaluate["
{
  freq <- 4
    st<- 1947
    realst <-
     tsp(ts(gnp[seq(1:before)],frequency=freq,start=c(st,2)))
     [[2]]+1/freq
    gnpinteractive <-
     ts(gnp[seq(before+1,before+width)],frequency=freq,start=
     c(realst,2))
}"];
  plot["
plot(gnpinteractive)
points(gnpinteractive,pch='*')"
  ]]
```

Creating an interface with `Manipulate` can be done immediately. Since this is done via an intermediate file, as soon as the picture meets the desired requirement, you can just grab that file; no exporting needed:

```
Manipulate[plotGnpZoomed[before, width],
 {before, 0, Length[gnp] - 1, 1},
 {{width, Length[gnp]}, 1, Length[gnp], 1}]
```



5. First, we fit the time series data to an autoregressive (AR) model using the maximum-likelihood estimation in R:

```
REvaluate["m1=ar(gnp,method=\"mle\")"] // Short
```

```
RObject({{3}, {0.342668, 0.141439, −0.131884}, ≪10≫, ≪1≫, (≪1≫)}, ≪1≫)
```

6. What is important in an AR model are its coefficients and order. Combining *Finance Platform*'s string operation with R's analysis, we present the AR model's coefficients, order, and standard error below. The coefficients show that the data is fitted using an AR(3) model.

```
Framed@StringJoin@Riffle[#, "\n"] &@
 REvaluate["capture.output(print(m1))"]
```

```
Call:
ar(x = gnp, method = "mle")

Coefficients:
     1        2        3
 0.3427   0.1414  −0.1319

Order selected 3  sigma^2 estimated as  8.436e−05
```

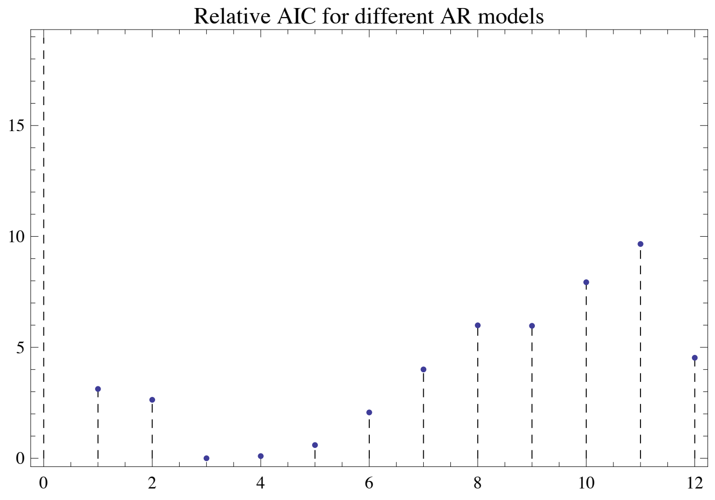
The Akaike information criterion (AIC) is used to verify the AR model's order. The following command gives the AIC differences for several different time lags:

```
aics =
  MapIndexed[{First[#2] - 1, #1} &, First@REvaluate["m1$aic"]];
```

7. We can visualize the result along with its tabular form. From the graph, it is easy to see that the optimal value of the time lag is 3:

```
Row[{TableForm[aics, TableHeadings → {None, {"Lag", "AIC diff"}}],
  Spacer[10], ListPlot[aics, PlotStyle → Thick, Filling → Axis,
   FillingStyle → Dashed, ImageSize → 400, Frame → True,
   PlotLabel → "Relative AIC for different AR models"]}]
```

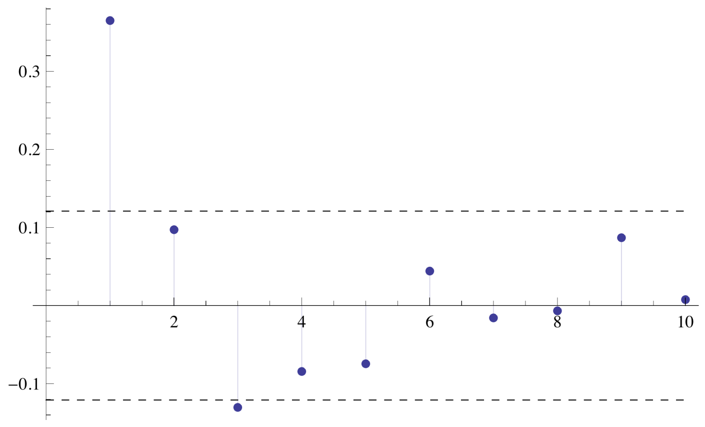
Lag	AIC diff
0	38.6016
1	3.12438
2	2.63561
3	0.
4	0.0974919
5	0.594544
6	2.06579
7	4.00438
8	5.99201
9	5.97041
10	7.93484
11	9.65893
12	4.53024



8. In time series analysis, the partial autocorrelation function (PACF) can be used to verify the extent of the lag in an autoregressive model. You can use *Finance Platform*'s built-in `PartialCorrelationFunction` to carry out this verification:

```
pacf[data_, lmax_, clev_: 0.95] :=
 Show[ListPlot[PartialCorrelationFunction[data, {lmax}],
   Filling → Axis, PlotStyle → PointSize[Medium],
   AxesOrigin → {0, 0}, PlotRange → All],
  Graphics[{Dashed, Line[{{0, #}, {lmax, #}}]}] & /@
   Quantile[NormalDistribution[], {(1 - clev)/2, 1 - (1 - clev)/2}] /
    (√Length[data])]
```

```
pacf[data, 10, .95]
```



9. Once we obtain the order of the AR process, an autoregressive integrated moving-average (ARIMA) model can be applied to find the business cycles:

```
REvaluate["m1$order"]
```

```
{3}
```

```
REvaluate["m2 <- arima(gnp,order=c(m1$order, 0, 0));"]
```

Here are the coefficients, and the intercept is found:

```
res = REvaluate["m2$coef"]
```

```
RObject({0.342664, 0.141463, -0.131929, 0.00789193},
  RAttributes(names :→ {ar1, ar2, ar3, intercept}))
```

`RObject` is a normal *Mathematica*® expression. We can extract the data as follows:

```
First@res
```

```
{0.342664, 0.141463, -0.131929, 0.00789193}
```

Now we are ready to find the period of the business cycle. By adding one as the zeroth order coefficient, we can construct a polynomial:

```
ClearAll[x];
coeffs = {1} ~ Join ~ - Most[First[res]];
poly = x^Range[0, First[REvaluate["m1$order"]]].coeffs
```

$0.131929\, x^3 - 0.141463\, x^2 - 0.342664\, x + 1$

Solving for its roots:

```
sol =
 NSolve[x^Range[0, First[REvaluate["m1$order"]]].coeffs == 0, x]
```

$\{\{x \to -2.03482\}, \{x \to 1.55354 - 1.14523\, i\}, \{x \to 1.55354 + 1.14523\, i\}\}$

Since the complex roots are responsible for the oscillations, we can choose one of them and find its frequency:

```
root = x /. sol[[2]]
```

$1.55354 - 1.14523\, i$

Frequency is then:

```
fr = Abs[Arg[root]]
```

```
0.63524
```

Find the cycle (period) in quarters:

```
period = 2 Pi / fr
```

```
9.89103
```

This is just about two and half years:

```
period / 4
```

```
2.47276
```

Pricing and Licensing Information

Wolfram *Finance Platform* includes all of the technologies and support services that you need to be productive.

Purchase levels scale from small teams to entire enterprise-wide deployment. Contact our finance industry experts to discuss a quote.

Visit us online for more information:
www.wolfram.com/finance-platform/contact-us

Recommended Next Steps

Watch videos about Wolfram *Finance Platform*
www.wolfram.com/broadcast/video.php?channel=249

Request a free trial or schedule a technical demo
www.wolfram.com/finance-platform/contact-us

Learn more about Wolfram *Finance Platform*

US and Canada
1-800-WOLFRAM (965-3726)
info@wolfram.com

Europe
+44-(0)1993-883400
info@wolfram.co.uk

Outside US and Canada (except Europe and Asia)
+1-217-398-0700
info@wolfram.com

Asia
+81-(0)3-3518-2880
info@wolfram.co.jp