**Acknowledgements**

The code in this iPython notebook used to be in R. I am very grateful to Yves Hilpisch and Michael Schwed for translating my R-code to Python.

For slideshow functionality I use RISE by Damián Avila.

**Outline of presentation**

* The time series of historical volatility
  + Scaling properties
* The RFSV model
* Pricing under rough volatility
* Forecasting realized variance
* The time series of variance swaps
* Relating historical and implied

**The time series of realized variance**

* Assuming an underlying variance process vs𝑣𝑠, integrated variance 1δ∫t+δtvsds1𝛿∫𝑡𝑡+𝛿𝑣𝑠𝑑𝑠 may (in principle) be estimated arbitrarily accurately given enough price data.
  + In practice, market microstructure noise makes estimation harder at very high frequency.
  + Sophisticated estimators of integrated variance have been developed to adjust for market microstructure noise. See Gatheral and Oomen [[6]](https://tpq.io/p/rough_volatility_with_python.html#cite_note-GO) (for example) for details of these.
* The Oxford-Man Institute of Quantitative Finance makes historical realized variance (RV) estimates freely available at [http://realized.oxford-man.ox.ac.uk](http://realized.oxford-man.ox.ac.uk/). These estimates are updated daily.
  + Each day, for 21 different indices, all trades and quotes are used to estimate realized (or integrated) variance over the trading day from open to close.
* Using daily RV estimates as proxies for instantaneous variance, we may investigate the time series properties of vt𝑣𝑡 empirically.

First load all necessary Python libraries.

In [1]:

**import** **warnings**; warnings.simplefilter('ignore')

**import** **datetime** **as** **dt**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib.mlab** **as** **mlab**

**from** **matplotlib.mlab** **import** stineman\_interp

**import** **pandas** **as** **pd**

**import** **pandas.io.data** **as** **web**

**import** **requests**

**import** **zipfile** **as** **zi**

**import** **StringIO** **as** **sio**

**from** **sklearn** **import** datasets, linear\_model

**import** **scipy.special** **as** **scsp**

**import** **statsmodels.api** **as** **sm**

**import** **math**

**import** **seaborn** **as** **sns**; sns.set()

%**matplotlib** inline

Then update and save the latest Oxford-Man data.

In [2]:

url = 'http://realized.oxford-man.ox.ac.uk/media/1366/'

url += 'oxfordmanrealizedvolatilityindices.zip'

data = requests.get(url, stream=True).content

z = zi.ZipFile(sio.StringIO(data))

z.extractall()

There are many different estimates of realized variance, all of them very similar. We will use the realized kernel estimates denoted by ".rk".

In [3]:

df = pd.read\_csv('OxfordManRealizedVolatilityIndices.csv', index\_col=0, header=2 )

rv1 = pd.DataFrame(index=df.index)

**for** col **in** df.columns:

**if** col[-3:] == '.rk':

rv1[col] = df[col]

rv1.index = [dt.datetime.strptime(str(date), "%Y%m**%d**") **for** date **in** rv1.index.values]

Let's plot SPX realized variance.

In [4]:

spx = pd.DataFrame(rv1['SPX2.rk'])

spx.plot(color='red', grid=True, title='SPX realized variance',

figsize=(16, 9), ylim=(0,0.003));

A graph showing a red line

Description automatically generated

Figure 1: Oxford-Man KRV estimates of SPX realized variance from January 2000 to the current date.

In [5]:

spx.head()

Out[5]:

|  | **SPX2.rk** |
| --- | --- |
| **2000-01-03** | 0.000161 |
| **2000-01-04** | 0.000264 |
| **2000-01-05** | 0.000305 |
| **2000-01-06** | 0.000149 |
| **2000-01-07** | 0.000123 |

In [6]:

spx.tail()

Out[6]:

|  | **SPX2.rk** |
| --- | --- |
| **2016-04-27** | 0.000031 |
| **2016-04-28** | 0.000031 |
| **2016-04-29** | 0.000048 |
| **2016-05-02** | 0.000028 |
| **2016-05-03** | 0.000041 |

We can get SPX data from Yahoo using the DataReader function:

In [7]:

SPX = web.DataReader(name = '^GSPC',data\_source = 'yahoo', start='2000-01-01')

SPX = SPX['Adj Close']

SPX.plot(title='SPX',figsize=(14, 8));

A graph showing a line

Description automatically generated with medium confidence

**The smoothness of the volatility process**

For q≥0𝑞≥0, we define the q𝑞th sample moment of differences of log-volatility at a given lag ΔΔ.(⟨⋅⟩⟨⋅⟩ denotes the sample average):

m(q,Δ)=⟨|logσt+Δ−logσt|q⟩𝑚(𝑞,Δ)=⟨|log⁡𝜎𝑡+Δ−log⁡𝜎𝑡|𝑞⟩

For example

m(2,Δ)=⟨(logσt+Δ−logσt)2⟩𝑚(2,Δ)=⟨(log⁡𝜎𝑡+Δ−log⁡𝜎𝑡)2⟩

is just the sample variance of differences in log-volatility at the lag ΔΔ.

**Scaling of**m(q,Δ)𝑚(𝑞,Δ)**with lag**ΔΔ

In [8]:

spx['sqrt']= np.sqrt(spx['SPX2.rk'])

spx['log\_sqrt'] = np.log(spx['sqrt'])

**def** del\_Raw(q, x):

**return** [np.mean(np.abs(spx['log\_sqrt'] - spx['log\_sqrt'].shift(lag)) \*\* q)

**for** lag **in** x]

In [9]:

plt.figure(figsize=(8, 8))

plt.xlabel('$log(\Delta)$')

plt.ylabel('$log\ m(q.\Delta)$')

plt.ylim=(-3, -.5)

zeta\_q = list()

qVec = np.array([.5, 1, 1.5, 2, 3])

x = np.arange(1, 100)

**for** q **in** qVec:

plt.plot(np.log(x), np.log(del\_Raw(q, x)), 'o')

model = np.polyfit(np.log(x), np.log(del\_Raw(q, x)), 1)

plt.plot(np.log(x), np.log(x) \* model[0] + model[1])

zeta\_q.append(model[0])

**print** zeta\_q

[0.072526605081197543, 0.14178030350291049, 0.20760692212873791, 0.27007948205423049, 0.38534332343872962]

A graph of colored lines

Description automatically generated

Figure 2: logm(q,Δ)log⁡𝑚(𝑞,Δ) as a function of logΔlog⁡Δ, SPX.

**Monofractal scaling result**

* From the above log-log plot, we see that for each q𝑞, m(q,Δ)∝Δζq𝑚(𝑞,Δ)∝Δ𝜁𝑞.
* How does ζq𝜁𝑞 scale with q𝑞?

**Scaling of**ζq𝜁𝑞**with**q𝑞

In [10]:

plt.figure(figsize=(8,8))

plt.xlabel('q')

plt.ylabel('$\zeta\_{q}$')

plt.plot(qVec, zeta\_q, 'or')

line = np.polyfit(qVec[:4], zeta\_q[:4],1)

plt.plot(qVec, line[0] \* qVec + line[1])

h\_est= line[0]

**print**(h\_est)

0.131697049909

A graph of a line

Description automatically generated

Figure 3: Scaling of ζq𝜁𝑞 with q𝑞.

We find the monofractal scaling relationship

ζq=qH𝜁𝑞=𝑞𝐻

with H≈0.13𝐻≈0.13.

* Note however that H𝐻 does vary over time, in a narrow range.
* Note also that our estimate of H𝐻 is biased high because we proxied instantaneous variance vt𝑣𝑡 with its average over each day 1T∫T0vtdt1𝑇∫0𝑇𝑣𝑡𝑑𝑡, where T𝑇 is one trading day.

**Estimated**H𝐻**for all indices**

We now repeat this analysis for all 21 indices in the Oxford-Man dataset.

In [11]:

**def** dlsig2(sic, x, pr=False):

**if** pr:

a= np.array([(sig-sig.shift(lag)).dropna() **for** lag **in** x])

a=a \*\* 2

**print** a.info()

**return** [np.mean((sig-sig.shift(lag)).dropna() \*\* 2) **for** lag **in** x]

In [12]:

h = list()

nu = list()

**for** col **in** rv1.columns:

sig = rv1[col]

sig = np.log(np.sqrt(sig))

sig = sig.dropna()

model = np.polyfit(np.log(x), np.log(dlsig2(sig, x)), 1)

nu.append(np.sqrt(np.exp(model[1])))

h.append(model[0]/2.)

OxfordH = pd.DataFrame({'names':rv1.columns, 'h\_est': h, 'nu\_est': nu})

In [13]:

OxfordH

Out[13]:

|  | **h\_est** | **names** | **nu\_est** |
| --- | --- | --- | --- |
| **0** | 0.133954 | SPX2.rk | 0.321337 |
| **1** | 0.142315 | FTSE2.rk | 0.270677 |
| **2** | 0.113366 | N2252.rk | 0.320396 |
| **3** | 0.150251 | GDAXI2.rk | 0.274873 |
| **4** | NaN | RUT2.rk | NaN |
| **5** | 0.083370 | AORD2.rk | 0.359025 |
| **6** | 0.131013 | DJI2.rk | 0.317327 |
| **7** | NaN | IXIC2.rk | NaN |
| **8** | 0.130485 | FCHI2.rk | 0.291967 |
| **9** | 0.103370 | HSI2.rk | 0.281453 |
| **10** | 0.125847 | KS11.rk | 0.274713 |
| **11** | 0.145671 | AEX.rk | 0.290187 |
| **12** | 0.178427 | SSMI.rk | 0.223249 |
| **13** | 0.128117 | IBEX2.rk | 0.281662 |
| **14** | 0.110092 | NSEI.rk | 0.324883 |
| **15** | 0.092252 | MXX.rk | 0.324180 |
| **16** | 0.106595 | BVSP.rk | 0.312907 |
| **17** | NaN | GSPTSE.rk | NaN |
| **18** | 0.119857 | STOXX50E.rk | 0.337045 |
| **19** | 0.127094 | FTSTI.rk | 0.228910 |
| **20** | 0.133361 | FTSEMIB.rk | 0.298712 |

**Distributions of**(logσt+Δ−logσt)(log⁡𝜎𝑡+Δ−log⁡𝜎𝑡)**for various lags**ΔΔ

Having established these beautiful scaling results for the moments, how do the histograms look?

In [14]:

**def** plotScaling(j, scaleFactor):

col\_name = rv1.columns[j]

v = rv1[col\_name]

x = np.arange(1,101)

**def** xDel(x, lag):

**return** x-x.shift(lag)

**def** sdl(lag):

**return** (xDel(np.log(v), lag)).std()

sd1 = (xDel(np.log(v), 1)).std()

h = OxfordH['h\_est'][j]

f, ax = plt.subplots(2,2,sharex=False, sharey=False, figsize=(10, 10))

**for** i\_0 **in** range(0, 2):

**for** i\_1 **in** range(0, 2):

la = scaleFactor \*\* (i\_1\*1+i\_0\*2)

hist\_val = xDel(np.log(v), la).dropna()

std = hist\_val.std()

mean = hist\_val.mean()

ax[i\_0][i\_1].set\_title('Lag = **%s** Days' %**la**)

n, bins, patches = ax[i\_0][i\_1].hist(hist\_val.values, bins=100,

normed=1, facecolor='green',alpha=0.2)

ax[i\_0][i\_1].plot(bins, mlab.normpdf(bins,mean,std), "r")

ax[i\_0][i\_1].plot(bins, mlab.normpdf(bins,0,sd1 \* la \*\* h), "b--")

hist\_val.plot(kind='density', ax=ax[i\_0][i\_1])

In [15]:

plotScaling(1,5)

A screenshot of a graph

Description automatically generated

Figure 4: Histograms of (logσt+Δ−logσt)(log⁡𝜎𝑡+Δ−log⁡𝜎𝑡) for various lags ΔΔ; normal fit in red; Δ=1Δ=1 normal fit scaled by Δ0.14Δ0.14 in blue.

**Universality?**

* [Gatheral, Jaisson and Rosenbaum][[5]</a></sup> compute daily realized variance estimates over one hour windows for DAX and Bund futures contracts, finding similar scaling relationships.](https://tpq.io/p/rough_volatility_with_python.html#cite_note-GJR)

[We have also checked that Gold and Crude Oil futures scale similarly.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GJR)

* + [Although the increments (logσt+Δ−logσt)(log⁡𝜎𝑡+Δ−log⁡𝜎𝑡) seem to be fatter tailed than Gaussian.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GJR)

[**A natural model of realized volatility**](https://tpq.io/p/rough_volatility_with_python.html#cite_note-GJR)

* As noted originally by [Andersen et al.][[1]</a></sup>, distributions of differences in the log of realized volatility are close to Gaussian.](https://tpq.io/p/rough_volatility_with_python.html#cite_note-ABDE)
  + [This motivates us to model σt𝜎𝑡 as a lognormal random variable.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-ABDE)
* [Moreover, the scaling property of variance of RV differences suggests the model:](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-ABDE)

(1)

logσt+Δ−logσt=ν(WHt+Δ−WHt)log⁡𝜎𝑡+Δ−log⁡𝜎𝑡=𝜈(𝑊𝑡+Δ𝐻−𝑊𝑡𝐻)

where WH𝑊𝐻 is fractional Brownian motion.

* In [Gatheral, Jaisson and Rosenbaum][[5]</a></sup>, we refer to a stationary version of](https://tpq.io/p/rough_volatility_with_python.html#cite_note-GJR)[(1)](https://tpq.io/p/rough_volatility_with_python.html#eq:dataDriven)as the RFSV (for Rough Fractional Stochastic Volatility) model.

**Fractional Brownian motion (fBm)**

* *Fractional Brownian motion* (fBm) {WHt;t∈R}{𝑊𝑡𝐻;𝑡∈𝑅} is the unique Gaussian process with mean zero and autocovariance function

E[WHtWHs]=12{|t|2H+|s|2H−|t−s|2H}𝐸[𝑊𝑡𝐻𝑊𝑠𝐻]=12{|𝑡|2𝐻+|𝑠|2𝐻−|𝑡−𝑠|2𝐻}

where H∈(0,1)𝐻∈(0,1) is called the *Hurst index* or parameter.

* + In particular, when H=1/2𝐻=1/2, fBm is just Brownian motion.
  + If H>1/2𝐻>1/2, increments are positively correlated.% so the process is trending.
  + If H<1/2𝐻<1/2, increments are negatively correlated.% so the process is reverting.

**Representations of fBm**

There are infinitely many possible representations of fBm in terms of Brownian motion. For example, with γ=12−H𝛾=12−𝐻,

**Mandelbrot-Van Ness**

WHt=CH{∫t−∞dWs(t−s)γ−∫0−∞dWs(−s)γ}.𝑊𝑡𝐻=𝐶𝐻{∫−∞𝑡𝑑𝑊𝑠(𝑡−𝑠)𝛾−∫−∞0𝑑𝑊𝑠(−𝑠)𝛾}.

The choice

CH=2HΓ(3/2−H)Γ(H+1/2)Γ(2−2H)−−−−−−−−−−−−−−−−−−−√𝐶𝐻=2𝐻Γ(3/2−𝐻)Γ(𝐻+1/2)Γ(2−2𝐻)

ensures that

E[WHtWHs]=12{t2H+s2H−|t−s|2H}.𝐸[𝑊𝑡𝐻𝑊𝑠𝐻]=12{𝑡2𝐻+𝑠2𝐻−|𝑡−𝑠|2𝐻}.

**Does simulated RSFV data look real?**

A blue lines with white text

Description automatically generated with medium confidenceFigure 8: Volatility of SPX (above) and of the RFSV model (below).

**Remarks on the comparison**

* The simulated and actual graphs look very alike.
* Persistent periods of high volatility alternate with low volatility periods.
* H∼0.1𝐻∼0.1 generates very rough looking sample paths (compared with H=1/2𝐻=1/2 for Brownian motion).
* Hence *rough volatility*.
* On closer inspection, we observe fractal-type behavior.
* The graph of volatility over a small time period looks like the same graph over a much longer time period.
* This feature of volatility has been investigated both empirically and theoretically in, for example, [Bacry and Muzy][[3]</a></sup> .](https://tpq.io/p/rough_volatility_with_python.html#cite_note-BacryMuzy)

[In particular, their Multifractal Random Walk (MRW) is related to a limiting case of the RSFV model as H→0𝐻→0.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BacryMuzy)

**[Pricing under rough volatility](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BacryMuzy)**

The foregoing behavior suggest the following model (see [Bayer et al.][[2]</a></sup> for volatility under the real (or historical or physical) measure P𝑃:](https://tpq.io/p/rough_volatility_with_python.html#cite_note-BayerFriz)

[logσt=νWHt.log⁡𝜎𝑡=𝜈𝑊𝑡𝐻.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

[Let γ=12−H𝛾=12−𝐻. We choose the Mandelbrot-Van Ness representation of fractional Brownian motion WH𝑊𝐻 as follows:](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

[WHt=CH{∫t−∞dWPs(t−s)γ−∫0−∞dWPs(−s)γ}.𝑊𝑡𝐻=𝐶𝐻{∫−∞𝑡𝑑𝑊𝑠𝑃(𝑡−𝑠)𝛾−∫−∞0𝑑𝑊𝑠𝑃(−𝑠)𝛾}.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

[Then](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

[==:logvu−logvtνCH{∫ut1(u−s)γdWPs+∫t−∞[1(u−s)γ−1(t−s)γ]dWPs}2νCH[Mt(u)+Zt(u)].log⁡𝑣𝑢−log⁡𝑣𝑡=𝜈𝐶𝐻{∫𝑡𝑢1(𝑢−𝑠)𝛾𝑑𝑊𝑠𝑃+∫−∞𝑡[1(𝑢−𝑠)𝛾−1(𝑡−𝑠)𝛾]𝑑𝑊𝑠𝑃}=:2𝜈𝐶𝐻[𝑀𝑡(𝑢)+𝑍𝑡(𝑢)].](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

[Note that EP[Mt(u)|Ft]=0𝐸𝑃[𝑀𝑡(𝑢)|𝐹𝑡]=0 and Zt(u)𝑍𝑡(𝑢) is Ft𝐹𝑡-measurable.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

* + [To price options, it would seem that we would need to know Ft𝐹𝑡, the entire history of the Brownian motion Ws𝑊𝑠 for $s<t$!< li="" style="box-sizing: border-box;"></t$!<>](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

**[Pricing under](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)**[P𝑃](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-BayerFriz)

Let

W~Pt(u):=2H−−−√∫utdWPs(u−s)γ𝑊~𝑡𝑃(𝑢):=2𝐻∫𝑡𝑢𝑑𝑊𝑠𝑃(𝑢−𝑠)𝛾

With η:=2νCH/2H−−−√𝜂:=2𝜈𝐶𝐻/2𝐻 we have 2νCHMt(u)=ηW~Pt(u)2𝜈𝐶𝐻𝑀𝑡(𝑢)=𝜂𝑊~𝑡𝑃(𝑢) so denoting the stochastic exponential by E(⋅)𝐸(⋅), we may write

vu==vtexp{ηW~Pt(u)+2νCHZt(u)}EP[vu|Ft]E(ηW~Pt(u)).𝑣𝑢=𝑣𝑡exp⁡{𝜂𝑊~𝑡𝑃(𝑢)+2𝜈𝐶𝐻𝑍𝑡(𝑢)}=𝐸𝑃[𝑣𝑢|𝐹𝑡]𝐸(𝜂𝑊~𝑡𝑃(𝑢)).

* The conditional distribution of vu𝑣𝑢 depends on Ft𝐹𝑡 only through the variance forecasts EP[vu|Ft]𝐸𝑃[𝑣𝑢|𝐹𝑡],
* To price options, one does not need to know Ft𝐹𝑡, the entire history of the Brownian motion WPs𝑊𝑠𝑃 for $s<t$.< li="" style="box-sizing: border-box;"></t$.<>

**Pricing under**Q𝑄

Our model under P𝑃 reads:

(2)

vu=EP[vu|Ft]E(ηW~Pt(u)).𝑣𝑢=𝐸𝑃[𝑣𝑢|𝐹𝑡]𝐸(𝜂𝑊~𝑡𝑃(𝑢)).

Consider some general change of measure

dWPs=dWQs+λsds,𝑑𝑊𝑠𝑃=𝑑𝑊𝑠𝑄+𝜆𝑠𝑑𝑠,

where {λs:s>t}{𝜆𝑠:𝑠>𝑡} has a natural interpretation as the price of volatility risk.

We may then rewrite [(2)](https://tpq.io/p/rough_volatility_with_python.html" \l "eq:Pmodel) as

vu=EP[vu|Ft]E(ηW~Qt(u))exp{η2H−−−√∫utλs(u−s)γds}.𝑣𝑢=𝐸𝑃[𝑣𝑢|𝐹𝑡]𝐸(𝜂𝑊~𝑡𝑄(𝑢))exp⁡{𝜂2𝐻∫𝑡𝑢𝜆𝑠(𝑢−𝑠)𝛾𝑑𝑠}.

* Although the conditional distribution of vu𝑣𝑢 under P𝑃 is lognormal, it will not be lognormal in general under Q𝑄.
  + The upward sloping smile in VIX options means λs𝜆𝑠 cannot be deterministic in this picture.

**The rough Bergomi (rBergomi) model**

Let's nevertheless consider the simplest change of measure

dWPs=dWQs+λ(s)ds,𝑑𝑊𝑠𝑃=𝑑𝑊𝑠𝑄+𝜆(𝑠)𝑑𝑠,

where λ(s)𝜆(𝑠) is a deterministic function of s𝑠. Then from [(2)](https://tpq.io/p/rough_volatility_with_python.html" \l "eq:Pmodel), we would have

vu==EP[vu|Ft]E(ηW~Qt(u))exp{η2H−−−√∫ut1(u−s)γλ(s)ds}ξt(u)E(ηW~Qt(u))𝑣𝑢=𝐸𝑃[𝑣𝑢|𝐹𝑡]𝐸(𝜂𝑊~𝑡𝑄(𝑢))exp⁡{𝜂2𝐻∫𝑡𝑢1(𝑢−𝑠)𝛾𝜆(𝑠)𝑑𝑠}=𝜉𝑡(𝑢)𝐸(𝜂𝑊~𝑡𝑄(𝑢))

where the forward variances ξt(u)=EQ[vu|Ft]𝜉𝑡(𝑢)=𝐸𝑄[𝑣𝑢|𝐹𝑡] are (at least in principle) tradable and observed in the market.

* ξt(u)𝜉𝑡(𝑢) is the product of two terms:
  + EP[vu|Ft]𝐸𝑃[𝑣𝑢|𝐹𝑡] which depends on the historical path $\{W\_s, s<="" li="" style="box-sizing: border-box;">
  + a term which depends on the price of risk λ(s)𝜆(𝑠).

**Features of the rough Bergomi model**

* The rBergomi model is a non-Markovian generalization of the Bergomi model:

E[vu|Ft]≠E[vu|vt].𝐸[𝑣𝑢|𝐹𝑡]≠𝐸[𝑣𝑢|𝑣𝑡].

* + The rBergomi model is Markovian in the (infinite-dimensional) state vector EQ[vu|Ft]=ξt(u)𝐸𝑄[𝑣𝑢|𝐹𝑡]=𝜉𝑡(𝑢).
* We have achieved our aim from Session 1 of replacing the exponential kernels in the Bergomi model with a power-law kernel.
* We may therefore expect that the rBergomi model will generate a realistic term structure of ATM volatility skew.

**Re-interpretation of the conventional Bergomi model**

* A conventional n𝑛-factor Bergomi model is not self-consistent for an arbitrary choice of the initial forward variance curve ξt(u)𝜉𝑡(𝑢).
  + ξt(u)=E[vu|Ft]𝜉𝑡(𝑢)=𝐸[𝑣𝑢|𝐹𝑡] should be consistent with the assumed dynamics.
* Viewed from the perspective of the fractional Bergomi model however:
  + The initial curve ξt(u)𝜉𝑡(𝑢) reflects the history $\{W\_s; s<="" li="" style="box-sizing: border-box;">
  + The exponential kernels in the exponent of the conventional Bergomi model approximate more realistic power-law kernels.
* The conventional two-factor Bergomi model is then justified in practice as a tractable Markovian engineering approximation to a more realistic fractional Bergomi model.

**The stock price process**

* The observed anticorrelation between price moves and volatility moves may be modeled naturally by anticorrelating the Brownian motion W𝑊 that drives the volatility process with the Brownian motion driving the price process.
* Thus

dStSt=vt−−√dZt𝑑𝑆𝑡𝑆𝑡=𝑣𝑡𝑑𝑍𝑡

with

dZt=ρdWt+1−ρ2−−−−−√dW⊥t𝑑𝑍𝑡=𝜌𝑑𝑊𝑡+1−𝜌2𝑑𝑊𝑡⊥

where ρ𝜌 is the correlation between volatility moves and price moves.

**Simulation of the rBergomi model**

We simulate the rBergomi model as follows:

* Construct the joint covariance matrix for the Volterra process W~𝑊~ and the Brownian motion Z𝑍 and compute its Cholesky decomposition.
* For each time, generate iid normal random vectors {and multiply them by the lower-triangular matrix obtained by the Cholesky decomposition} to get a m×2n𝑚×2𝑛 matrix of paths of W~𝑊~ and Z𝑍 with the correct joint marginals.
* With these paths held in memory, we may evaluate the expectation under Q𝑄 of any payoff of interest.
* This procedure is very slow!
  + Speeding up the simulation is work in progress.

**Guessing rBergomi model parameters**

* The rBergomi model has only three parameters: H𝐻, η𝜂 and ρ𝜌.
* If we had a fast simulation, we could just iterate on these parameters to find the best fit to observed option prices. But we don't.
* However, the model parameters H𝐻, η𝜂 and ρ𝜌 have very direct interpretations:
  + H𝐻 controls the decay of ATM skew ψ(τ)𝜓(𝜏) for very short expirations.
  + The product ρη𝜌𝜂 sets the level of the ATM skew for longer expirations.
    - Keeping ρη𝜌𝜂 constant but decreasing ρ𝜌 (so as to make it more negative) pushes the minimum of each smile towards higher strikes.
* So we can guess parameters in practice.
* As we will see, even without proper calibration (*i.e.* just guessing parameters), rBergomi model fits to the volatility surface are amazingly good.

**SPX smiles in the rBergomi model**

* In Figures 9 and 10, we show how well a rBergomi model simulation with guessed parameters fits the SPX option market as of February 4, 2010, a day when the ATM volatility term structure happened to be pretty flat.
  + rBergomi parameters were: H=0.07𝐻=0.07, η=1.9𝜂=1.9, ρ=−0.9𝜌=−0.9.
* Only three parameters to get a very good fit to the whole SPX volatility surface!

**rBergomi fits to SPX smiles as of 04-Feb-2010**

A group of graphs showing the same number of numbers

Description automatically generated with medium confidenceFigure 9: Red and blue points represent bid and offer SPX implied volatilities; orange smiles are from the rBergomi simulation.

**Shortest dated smile as of February 4, 2010**

A graph of a graph

Description automatically generated with medium confidenceFigure 10: Red and blue points represent bid and offer SPX implied volatilities; orange smile is from the rBergomi simulation.

**ATM volatilities and skews**

In Figures 11 and 12, we see just how well the rBergomi model can match empirical skews and vols. Recall also that the parameters we used are just guesses!

**Term structure of ATM skew as of February 4, 2010**

A red line with blue dots

Description automatically generatedFigure 11: Blue points are empirical skews; the red line is from the rBergomi simulation.

**Term structure of ATM vol as of February 4, 2010**

A graph with red and blue dots

Description automatically generatedFigure 12: Blue points are empirical ATM volatilities; the red line is from the rBergomi simulation.

**Another date**

* Now we take a look at another date: August 14, 2013, two days before the last expiration date in our dataset.
* Options set at the open of August 16, 2013 so only one trading day left.
* Note in particular that the extreme short-dated smile is well reproduced by the rBergomi model.
* There is no need to add jumps!

**SPX smiles as of August 14, 2013**

A group of graphs showing the same size

Description automatically generated with medium confidenceFigure 13: Red and blue points represent bid and offer SPX implied volatilities; orange smiles are from the rBergomi simulation.

**The forecast formula**

* In the RFSV model [(1)](https://tpq.io/p/rough_volatility_with_python.html#eq:dataDriven), logσt≈νWHt+Clog⁡𝜎𝑡≈𝜈𝑊𝑡𝐻+𝐶 for some constant C𝐶.
* [Nuzman and Poor][[7]</a></sup> show that WHt+Δ𝑊𝑡+Δ𝐻 is conditionally Gaussian with conditional expectation](https://tpq.io/p/rough_volatility_with_python.html#cite_note-NuzmanPoor)

[E[WHt+Δ|Ft]=cos(Hπ)πΔH+1/2∫t−∞WHs(t−s+Δ)(t−s)H+1/2ds𝐸[𝑊𝑡+Δ𝐻|𝐹𝑡]=cos⁡(𝐻𝜋)𝜋Δ𝐻+1/2∫−∞𝑡𝑊𝑠𝐻(𝑡−𝑠+Δ)(𝑡−𝑠)𝐻+1/2𝑑𝑠](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-NuzmanPoor)

[and conditional variance](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-NuzmanPoor)

[Var[WHt+Δ|Ft]=cΔ2H.Var[𝑊𝑡+Δ𝐻|𝐹𝑡]=𝑐Δ2𝐻.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-NuzmanPoor)

[where](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-NuzmanPoor)

[c=Γ(3/2−H)Γ(H+1/2)Γ(2−2H).𝑐=Γ(3/2−𝐻)Γ(𝐻+1/2)Γ(2−2𝐻).](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-NuzmanPoor)

**[The forecast formula](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-NuzmanPoor)**

Thus, we obtain

**Variance forecast formula**

(3)

EP[vt+Δ|Ft]=exp{EP[log(vt+Δ)|Ft]+2cν2Δ2H}𝐸𝑃[𝑣𝑡+Δ|𝐹𝑡]=exp⁡{𝐸𝑃[log⁡(𝑣𝑡+Δ)|𝐹𝑡]+2𝑐𝜈2Δ2𝐻}

where

EP[logvt+Δ|Ft]=cos(Hπ)πΔH+1/2∫t−∞logvs(t−s+Δ)(t−s)H+1/2ds.𝐸𝑃[log⁡𝑣𝑡+Δ|𝐹𝑡]=cos⁡(𝐻𝜋)𝜋Δ𝐻+1/2∫−∞𝑡log⁡𝑣𝑠(𝑡−𝑠+Δ)(𝑡−𝑠)𝐻+1/2𝑑𝑠.

**Implement variance forecast in Python**

In [16]:

**def** c\_tilde(h):

**return** scsp.gamma(3. / 2. - h) / scsp.gamma(h + 1. / 2.) \* scsp.gamma(2. - 2. \* h)

**def** forecast\_XTS(rvdata, h, date, nLags, delta, nu):

i = np.arange(nLags)

cf = 1./((i + 1. / 2.) \*\* (h + 1. / 2.) \* (i + 1. / 2. + delta))

ldata = rvdata.truncate(after=date)

l = len(ldata)

ldata = np.log(ldata.iloc[l - nLags:])

ldata['cf'] = np.fliplr([cf])[0]

*# print ldata*

ldata = ldata.dropna()

fcst = (ldata.iloc[:, 0] \* ldata['cf']).sum() / sum(ldata['cf'])

**return** math.exp(fcst + 2 \* nu \*\* 2 \* c\_tilde(h) \* delta \*\* (2 \* h))

**SPX actual vs forecast variance**

In [17]:

rvdata = pd.DataFrame(rv1['SPX2.rk'])

nu = OxfordH['nu\_est'][0] *# Vol of vol estimate for SPX*

h = OxfordH['h\_est'][0]

n = len(rvdata)

delta = 1

nLags = 500

dates = rvdata.iloc[nLags:n-delta].index

rv\_predict = [forecast\_XTS(rvdata, h=h, date=d, nLags=nLags,

delta=delta, nu=nu) **for** d **in** dates]

rv\_actual = rvdata.iloc[nLags+delta:n].values

**Scatter plot of delta days ahead predictions**

In [18]:

plt.figure(figsize=(8, 8))

plt.plot(rv\_predict, rv\_actual, 'bo');

A screen shot of a graph

Description automatically generated

Figure 14: Actual vols vs predicted vols.

**Superimpose actual and predicted vols**

In [19]:

plt.figure(figsize=(11, 6))

vol\_actual = np.sqrt(np.multiply(rv\_actual,252))

vol\_predict = np.sqrt(np.multiply(rv\_predict,252))

plt.plot(vol\_actual, "b")

plt.plot(vol\_predict, "r");

A graph showing a red and blue line

Description automatically generated

Figure 15: Actual volatilities in blue; predicted vols in red.

**Forecasting the variance swap curve**

Finally, we forecast the whole variance swap curve using the variance forecasting formula [(3)](https://tpq.io/p/rough_volatility_with_python.html#eq:vForecast).

In [20]:

**def** xi(date, tt, nu,h, tscale): *# dt=(u-t) is in units of years*

rvdata = pd.DataFrame(rv1['SPX2.rk'])

**return** [ forecast\_XTS(rvdata,h=h,date=date,nLags=500,delta=dt\*tscale,nu=nu) **for** dt **in** tt]

nu = OxfordH["nu\_est"][0]

h = OxfordH["h\_est"][0]

**def** varSwapCurve(date, bigT, nSteps, nu, h, tscale, onFactor):

*# Make vector of fwd variances*

tt = [ float(i) \* (bigT) / nSteps **for** i **in** range(nSteps+1)]

delta\_t = tt[1]

xicurve = xi(date, tt, nu, h, tscale)

xicurve\_mid = (np.array(xicurve[0:nSteps]) + np.array(xicurve[1:nSteps+1])) / 2

xicurve\_int = np.cumsum(xicurve\_mid) \* delta\_t

varcurve1 = np.divide(xicurve\_int, np.array(tt[1:]))

varcurve = np.array([xicurve[0],]+list(varcurve1))

varcurve = varcurve \* onFactor \* tscale *# onFactor is to compensate for overnight moves*

res = pd.DataFrame({"texp":np.array(tt), "vsQuote":np.sqrt(varcurve)})

**return**(res)

In [21]:

**def** varSwapForecast(date,tau,nu,h,tscale,onFactor):

vsc = varSwapCurve(date, bigT=2.5, nSteps=100, nu=nu, h=h,

tscale=tscale, onFactor=onFactor) *# Creates the whole curve*

x = vsc['texp']

y = vsc['vsQuote']

res = stineman\_interp(tau,x,y,None)

**return**(res)

*# Test the function*

tau = (.25,.5,1,2)

date = dt.datetime(2008,9,8)

varSwapForecast(date,tau,nu=nu,h=h,tscale=252,onFactor=1)

Out[21]:

array([ 0.21949454, 0.21398188, 0.2117466 , 0.21262899])

**'Constructing a time series of variance swap curves**

For each of 2,658 days from Jan 27, 2003 to August 31, 2013:

* We compute proxy variance swaps from closing prices of SPX options sourced from OptionMetrics (www.optionmetrics.com) via WRDS.
* We form the forecasts EP[vu|Ft]𝐸𝑃[𝑣𝑢|𝐹𝑡] using [(3)](https://tpq.io/p/rough_volatility_with_python.html#eq:vForecast) with 500 lags of SPX RV data sourced from The Oxford-Man Institute of Quantitative Finance ([http://realized.oxford-man.ox.ac.uk](http://realized.oxford-man.ox.ac.uk/)).
* We note that the actual variance swap curve is a factor (of roughly 1.4) higher than the forecast, which we may attribute to a combination of overnight movements of the index and the price of volatility risk.
* Forecasts must therefore be rescaled to obtain close-to-close realized variance forecasts.

**3-month forecast vs actual variance swaps**

A graph showing the growth of a company

Description automatically generated with medium confidenceFigure 16: Actual (proxy) 3-month variance swap quotes in blue vs forecast in red (with no scaling factor).

**Ratio of actual to forecast**

A green line graph with numbers

Description automatically generatedFigure 17: The ratio between 3-month actual variance swap quotes and 3-month forecasts.

**The Lehman weekend**

* Empirically, it seems that the variance curve is a simple scaling factor times the forecast, but that this scaling factor is time-varying.
  + We can think of this factor as having two multiplicative components: the overnight factor, and the price of volatility risk.
* Recall that as of the close on Friday September 12, 2008, it was widely believed that Lehman Brothers would be rescued over the weekend. By Monday morning, we knew that Lehman had failed.
* In Figure 18, we see that variance swap curves just before and just after the collapse of Lehman are just rescaled versions of the RFSV forecast curves.

**We need variance swap estimates for 12-Sep-2008 and 15-Sep-2008**

We proxy these by taking SVI fits for the two dates and computing the log-strips.

In [22]:

varSwaps12 =(

0.2872021, 0.2754535, 0.2601864, 0.2544684, 0.2513854, 0.2515314,

0.2508418, 0.2520099, 0.2502763, 0.2503309, 0.2580933, 0.2588361,

0.2565093)

texp12 = (

0.01916496, 0.04654346, 0.09582478, 0.19164956, 0.26830938, 0.29842574,

0.51745380, 0.54483231, 0.76659822, 0.79397673, 1.26488706, 1.76317591,

2.26146475)

varSwaps15 = (

0.4410505, 0.3485560, 0.3083603, 0.2944378, 0.2756881, 0.2747838,

0.2682212, 0.2679770, 0.2668113, 0.2706713, 0.2729533, 0.2689598,

0.2733176)

texp15 = (

0.01095140, 0.03832991, 0.08761123, 0.18343600, 0.26009582, 0.29021218,

0.50924025, 0.53661875, 0.75838467, 0.78576318, 1.25667351, 1.75496235,

2.25325120)

**Actual vs predicted over the Lehman weekend**

In [23]:

nu = OxfordH['nu\_est'][0]

h = OxfordH['h\_est'][0]

date1 = dt.datetime(2008, 9, 12)

date2 = dt.datetime(2008, 9, 15)

*# Variance curve fV model forecasts*

tau1000 = [ float(i) \* 2.5 / 1000. **for** i **in** range(1,1001)]

vs1 = varSwapForecast(date1, tau1000, nu=nu,h=h, tscale=252, onFactor=1.29)

vs2 = varSwapForecast(date2, tau1000, nu=nu,h=h, tscale=252, onFactor=1.29)

In [24]:

plt.figure(figsize=(11, 6))

plt.plot(texp12, varSwaps12, "r")

plt.plot(texp15, varSwaps15, "b")

plt.plot(tau1000, vs1, "r--")

plt.plot(tau1000, vs2, "b--");

A graph showing the results of a graph

Description automatically generated

Figure 18: SPX variance swap curves as of September 12, 2008 (red) and September 15, 2008 (blue). The dashed curves are RFSV model forecasts rescaled by the 3-month ratio (1.291.29) as of the Friday close.

**Remarks**

We note that

* The actual variance swaps curves are very close to the forecast curves, up to a scaling factor.
* We are able to explain the change in the variance swap curve with only one extra observation: daily variance over the trading day on Monday 15-Sep-2008.
* The SPX options market appears to be backward-looking in a very sophisticated way.

**The Flash Crash**

* The so-called Flash Crash of Thursday May 6, 2010 caused intraday realized variance to be much higher than normal.
* In Figure 19, we plot the actual variance swap curves as of the Wednesday and Friday market closes together with forecast curves rescaled by the 3-month ratio as of the close on Wednesday May 5 (which was 2.522.52).
* We see that the actual variance curve as of the close on Friday is consistent with a forecast from the time series of realized variance that *includes* the anomalous price action of Thursday May 6.

**Variance swap estimates**

We again proxy variance swaps for 05-May-2010, 07-May-2010 and 10-May-2010 by taking SVI fits (see [Gatheral and Jacquier][[4]</a></sup> ) for the three dates and computing the log-strips.](https://tpq.io/p/rough_volatility_with_python.html#cite_note-GatheralJacquierSSVI)

[In [25]:](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[varSwaps5 = (](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.4250369, 0.2552473, 0.2492892, 0.2564899, 0.2612677, 0.2659618, 0.2705928, 0.2761203,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.2828139, 0.2841165, 0.2884955, 0.2895839, 0.2927817, 0.2992602, 0.3116500)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[texp5 = (](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.002737851, 0.043805613, 0.120465435, 0.150581793, 0.197125257, 0.292950034,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.369609856, 0.402464066, 0.618754278, 0.654346338, 0.867898700, 0.900752909,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[1.117043121, 1.615331964, 2.631074606)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[varSwaps7 = (](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.5469727, 0.4641713, 0.3963352, 0.3888213, 0.3762354, 0.3666858, 0.3615814, 0.3627013,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.3563324, 0.3573946, 0.3495730, 0.3533829, 0.3521515, 0.3506186, 0.3594066)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[texp7 = (](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.01642710, 0.03832991, 0.11498973, 0.14510609, 0.19164956, 0.28747433, 0.36413415,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.39698836, 0.61327858, 0.64887064, 0.86242300, 0.89527721, 1.11156742, 1.60985626,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[2.62559890)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[varSwaps10 = (](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.3718439, 0.3023223, 0.2844810, 0.2869835, 0.2886912, 0.2905637, 0.2957070, 0.2960737,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.3005086, 0.3031188, 0.3058492, 0.3065815, 0.3072041, 0.3122905, 0.3299425)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[texp10 = (](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.008213552, 0.030116359, 0.106776181, 0.136892539, 0.183436003, 0.279260780,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[0.355920602, 0.388774812, 0.605065024, 0.640657084, 0.854209446, 0.887063655,](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[1.103353867, 1.601642710, 2.617385352)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[In [26]:](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[date1 = dt.datetime(2010, 5, 5)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[date2 = dt.datetime(2010, 5, 7)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[vsf5 = varSwapCurve(date1, bigT=2.5, nSteps=100, nu=nu, h=h, tscale=252, onFactor=2.52)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[vsf7 = varSwapCurve(date2, bigT=2.5, nSteps=100, nu=nu, h=h, tscale=252, onFactor=2.52)](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[In [27]:](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.figure(figsize=(11, 6))](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.plot(texp5, varSwaps5, "r", label='May 5')](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.plot(texp7, varSwaps7, "g", label='May 7')](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.legend()](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.xlabel("Time to maturity")](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.ylabel("Variance swap quote")](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.plot(vsf5['texp'], vsf5['vsQuote'], "r--")](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[plt.plot(vsf7['texp'], vsf7['vsQuote'], "g--");](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[A graph showing a number of different colored lines

Description automatically generated](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[Figure 19: SPX variance swap curves as of May 5, 2010 (red) and May 7, 2010 (green). The dashed curves are RFSV model forecasts rescaled by the 3-month ratio (2.522.52) as of the close on Wednesday May 5. The curve as of the close on May 7 is consistent with the forecast](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)**[including](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)**[the crazy moves on May 6.](https://tpq.io/p/rough_volatility_with_python.html" \l "cite_note-GatheralJacquierSSVI)

[**The weekend after the Flash Crash**](https://tpq.io/p/rough_volatility_with_python.html#cite_note-GatheralJacquierSSVI)

Now we plot forecast and actual variance swap curves as of the close on Friday May 7 and Monday May 10.

In [28]:

date1 = dt.datetime(2010,5,7)

date2 = dt.datetime(2010,5,10)

vsf7 = varSwapCurve(date1, bigT=2.5, nSteps=100, nu=nu, h=h, tscale=252, onFactor=2.52)

vsf10 = varSwapCurve(date2, bigT=2.5, nSteps=100, nu=nu, h=h, tscale=252, onFactor=2.52)

In [29]:

plt.figure(figsize=(11, 6))

plt.plot(texp7, varSwaps7, "g", label='May 7')

plt.plot(texp10, varSwaps10, "m", label='May 10')

plt.legend()

plt.xlabel("Time to maturity")

plt.ylabel("Variance swap quote")

plt.plot(vsf7['texp'], vsf7['vsQuote'], "g--")

plt.plot(vsf10['texp'], vsf10['vsQuote'], "m--");

A graph showing a line graph

Description automatically generated with medium confidence

Figure 20: The May 10 actual curve is inconsistent with a forecast that includes the Flash Crash.

Now let's see what happens if we exclude the Flash Crash from the time series used to generate the variance curve forecast.

In [30]:

plt.figure(figsize=(11, 6))

ax = plt.subplot(111)

rvdata\_p = rvdata.drop((dt.datetime(2010, 5, 6)), axis=0)

rvdata.loc["2010-05-04":"2010-05-10"].plot(ax=ax, legend=False)

rvdata\_p.loc["2010-05-04":"2010-05-10"].plot(ax=ax, legend=False);

A graph with a line in the middle

Description automatically generated

Figure 21: rvdata\_p has the May 6 realized variance datapoint eliminated (green line). Notice the crazy realized variance estimate for May 6!

We need a new variance curve forecast function that uses the new time series.

In [31]:

**def** xip(date, tt, nu,h, tscale): *# dt=(u-t) is in units of years*

rvdata = pd.DataFrame(rv1['SPX2.rk'])

rvdata\_p = rvdata.drop((dt.datetime(2010, 5, 6)), axis=0)

**return** [ forecast\_XTS(rvdata\_p, h=h, date=date,nLags=500,

delta=delta\_t \* tscale, nu=nu) **for** delta\_t **in** tt]

nu = OxfordH["nu\_est"][0]

h = OxfordH["h\_est"][0]

**def** varSwapCurve\_p(date, bigT, nSteps, nu, h, tscale, onFactor):

*# Make vector of fwd variances*

tt = [ float(i) \* (bigT) / nSteps **for** i **in** range(nSteps+1)]

delta\_t = tt[1]

xicurve = xip(date, tt, nu, h, tscale)

xicurve\_mid = (np.array(xicurve[0:nSteps]) + np.array(xicurve[1:nSteps + 1])) / 2

xicurve\_int = np.cumsum(xicurve\_mid) \* delta\_t

varcurve1 = np.divide(xicurve\_int, np.array(tt[1:]))

varcurve = np.array([xicurve[0],]+list(varcurve1))

varcurve = varcurve \* onFactor \* tscale *# onFactor is to compensate for overnight moves*

res = pd.DataFrame({"texp":np.array(tt), "vsQuote":np.sqrt(varcurve)})

**return**(res)

**def** varSwapForecast\_p(date, tau, nu, h, tscale, onFactor):

vsc = varSwapCurve\_p(date, bigT=2.5, nSteps=100, nu=nu, h=h,

tscale=tscale, onFactor=onFactor) *# Creates the whole curve*

x = vsc['texp']

y = vsc['vsQuote']

res = stineman\_interp(tau, x, y, None)

**return**(res)

*# Test the function*

tau = (.25, .5 ,1, 2)

date = dt.datetime(2010, 5, 10)

varSwapForecast\_p(date, tau, nu=nu, h=h, tscale=252, onFactor=1. / (1 - .35))

Out[31]:

array([ 0.26077084, 0.25255902, 0.25299844, 0.26116175])

Finally, we compare our new forecast curves with the actuals.

In [32]:

date1 = dt.datetime(2010, 5, 7)

date2 = dt.datetime(2010, 5, 10)

vsf7 = varSwapCurve(date1, bigT=2.5, nSteps=100, nu=nu, h=h, tscale=252, onFactor=2.52)

vsf10p = varSwapCurve\_p(date2, bigT=2.5, nSteps=100, nu=nu, h=h, tscale=252, onFactor=2.52)

In [33]:

plt.figure(figsize=(11, 6))

plt.plot(texp7, varSwaps7, "g", label='May 7')

plt.plot(texp10, varSwaps10, "m", label='May 10')

plt.legend()

plt.xlabel("Time to maturity")

plt.ylabel("Variance swap quote")

plt.plot(vsf7['texp'], vsf7['vsQuote'], "g--")

plt.plot(vsf10p['texp'], vsf10p['vsQuote'], "m--");

A graph showing a line of different colored lines

Description automatically generated with medium confidence

Figure 22: The May 10 actual curve is consistent with a forecast that excludes the Flash Crash.

**Resetting of expectations over the weekend**

* In Figures 20 and 22, we see that the actual variance swap curve on Monday, May 10 is consistent with a forecast that excludes the Flash Crash.
* Volatility traders realized that the Flash Crash should not influence future realized variance projections.

**Summary**

* We uncovered a remarkable monofractal scaling relationship in historical volatility.
  + A corollary is that volatility is not a long memory process, as widely believed.
* This leads to a natural non-Markovian stochastic volatility model under P𝑃.
* The simplest specification of dQdP𝑑𝑄𝑑𝑃 gives a non-Markovian generalization of the Bergomi model.
* The history of the Brownian motion $\lbrace W\_s, s<="" li="" style="box-sizing: border-box;">
* This model fits the observed volatility surface surprisingly well with very few parameters.
* For perhaps the first time, we have a simple consistent model of historical and implied volatility.

**References**

* 1. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-ABDE) Torben G Andersen, Tim Bollerslev, Francis X Diebold, and Heiko Ebens, The distribution of realized stock return volatility, \*Journal of Financial Economics\* \*\*61\*\*(1) 43-76 (2001).
  2. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-BayerFriz) Christian Bayer, Peter Friz and Jim Gatheral, Pricing under rough volatility, \*Quantitative Finance\* forthcoming, available at http://papers.ssrn.com/sol3/papers.cfm?abstract\_id=2554754, (2015).
  3. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-BacryMuzy) Emmanuel Bacry and Jean-François Muzy, Log-infinitely divisible multifractal processes, \*Communications in Mathematical Physics\* \*\*236\*\*(3) 449-475 (2003).
  4. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-GatheralJacquierSSVI) Jim Gatheral and Antoine Jacquier, Arbitrage-free SVI volatility surfaces, \*Quantitative Finance\* \*\*14\*\*(1) 59-71 (2014).
  5. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-GJR) Jim Gatheral, Thibault Jaisson and Mathieu Rosenbaum, Volatility is rough, available at http://papers.ssrn.com/sol3/papers.cfm?abstract\_id=2509457, (2014).
  6. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-GO) Jim Gatheral and Roel Oomen, Zero-intelligence realized variance estimation, \*Finance and Stochastics\* \*\*14\*\*(2) 249-283 (2010).
  7. [**^**](https://tpq.io/p/rough_volatility_with_python.html#cite_ref-NuzmanPoor) Carl J. Nuzman and H. Vincent Poor, Linear estimation of self-similar processes via Lamperti’s transformation, \*Journal of Applied Probability\* \*\*37\*\*(2) 429-452 (2000).