Relative cost of matrix factorizations

In [1]:

```
#keep
import numpy as np
import numpy.linalg as npla
import scipy.linalg as spla

import matplotlib.pyplot as pt
%matplotlib inline

from time import time
```

In [2]:

```
#keep
n_values = (10**np.linspace(1, 3.25, 15)).astype(np.int32)
n_values
```

Out[2]:

```
array([  10,   14,   20,   30,   43,   63,   92,  133,  193,  279,
404,
        585,  848, 1228, 1778], dtype=int32)
```

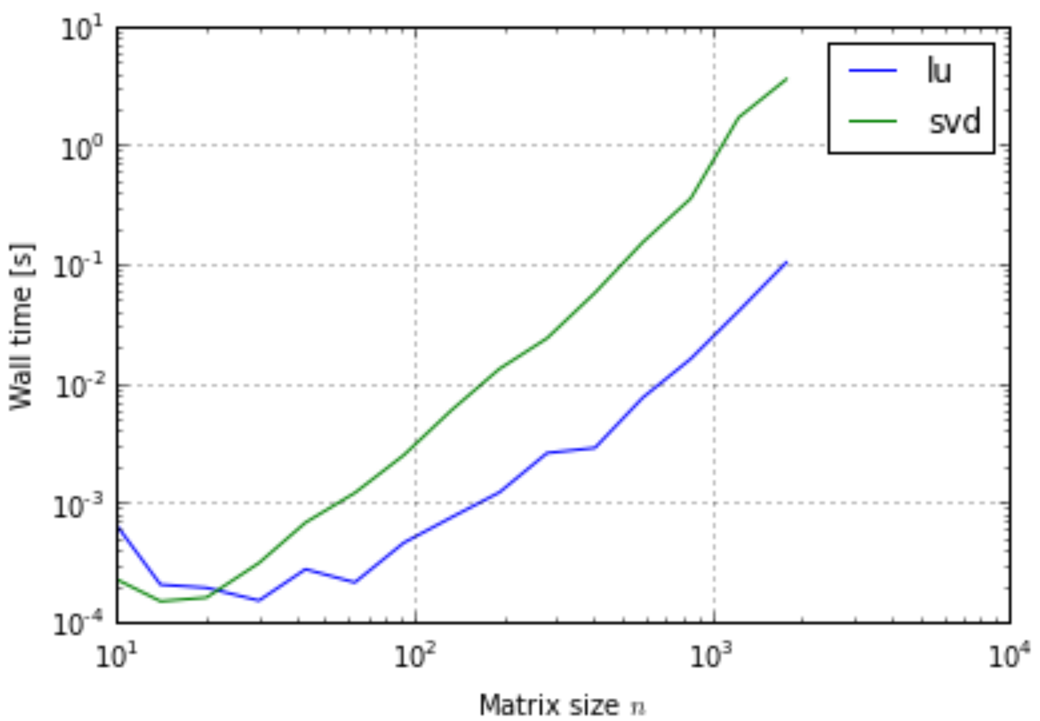
In [3]:

```
#keep
for name, f in [
        ("lu", spla.lu_factor),
        ("svd", npla.svd)
        ]:

    times = []
    print("----->", name)

    for n in n_values:
        print(n)

        A = np.random.randn(n, n)

        start_time = time()
        f(A)
        times.append(time() - start_time)

    pt.plot(n_values, times, label=name)

pt.grid()
pt.legend(loc="best")
pt.xlabel("Matrix size $n$")
pt.ylabel("Wall time [s]")
```

```
-----> lu
10
14
20
30
43
63
92
133
193
279
404
585
848
1228
1778
-----> svd
10
14
20
30
43
63
92
133
193
279
404
585
848
1228
1778
```

Out[3]:

```
<matplotlib.text.Text at 0x10d6b8d68>
```



- The faster algorithms make the slower ones look bad. But... it's all relative.
- Is there a better way of plotting this?
- Can we see the asymptotic cost ($O(n^3)$) of these algorithms from the plot?