In [57]:
```
%pylab inline
```

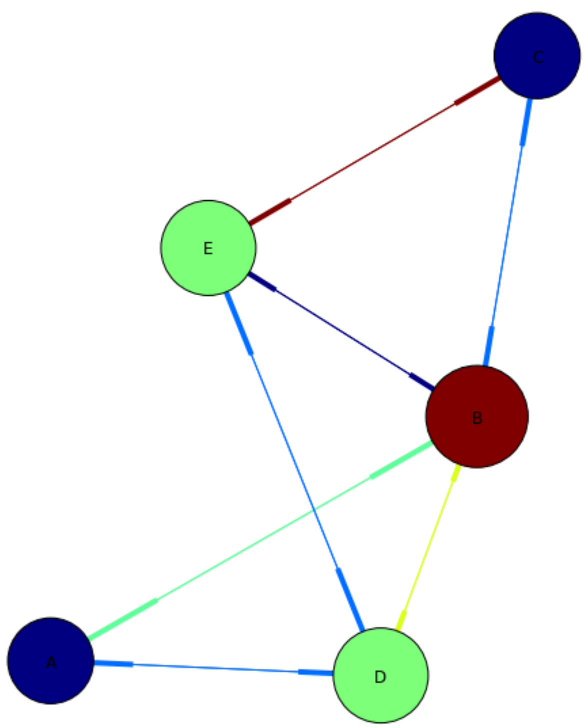
```
Welcome to pylab, a matplotlib-based Python environment [backend:
module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

Traffic Matrix Assignment

The following network topology is composed of 5 points of presence (you can also think of them as routers) and 7 bi-directional links (in blue) with associated weights (symmetrical for the pairs of links). The edge links are also depicted (in red).

In [58]:
```
print_link_legends()
cn.plot_graph(nx.degree,nx.degree_centrality,nx.draw_graphviz,G,edgecolors,labels)
```

```
5: Red
4: Yellow
3: Green
2 Light Blue
1 Dark Blue
```



The traffic matrix (somehow measured with Netflow, for example) for a certain time period is:

In [59]:
```
df_trafic
```

Out[59]:

	A	B	C	D	E
A	15	34	12	8	24
B	4	44	34	24	12
C	16	54	23	45	20
D	20	20	5	23	13
E	43	34	34	23	15

NOTE: We will use this data to practice several concepts. From the traffic matrix and the routing you will compute the link loads (external and internal), the gravity model, and the tomogravity solution. Of course, in a real situation where an operator wants to estimate the tomogravity model of the TM you start with the link loads (which is what you can easily measure with SNMP) and do not have the original, real traffic matrix; but in this way the exercise covers all the concepts, and you can even compare the tomogravity estimation with the original TM.

Type *Markdown* and LaTeX: α^2

a) Indicate the equation that relates the three following elements: link load vector, traffic matrix (in vector form), routing matrix. Indicate their dimensions.

In [60]:
```
Math(r'\mathbf{x}=\mathbf{A}·\mathbf{v}')
```

Out[60]: $\mathbf{x}=\mathbf{A}\cdot\mathbf{v}$

where x is the nx1 link load vector (n is the unmber of nodes), A is the nxr traffic matrix (r is the number of routes) and v is the n^2x1 trafic matrix vector

b) Compute the routing matrix. Is there any case where Equal Cost Multipath (ECMP) can be a possibility? How would you treat it?

In [61]:
```
df_route=tmo.route_matrix(G,ls_nodes,ls_edges)
pd.set_option('display.max_rows', 25)
pd.set_option('display.max_columns', 25)
df_route
```

Out[61]:

	Route	(A, A)	(A, B)	(A, C)	(A, D)	(A, E)	(B, A)	(B, B)	(B, C)	(B, D)	(B, E)	(C, A)	(C, B)	(C, C)	(C, D)	(C, E)	(D, A)	(D, B)	(D, C)	(D, D)	(D, E)	(E, A)	(E, B)	(E, C)	(E, D)	(E, E)
A	**D**	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	**A**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0
B	**A**	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
	C	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
C	**B**	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	**C**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0
	D	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0

d) Compute the internal link loads.

In [62]:
```
vec=df_trafic.values.reshape((size(df_trafic), 1))
load=np.dot(df_route.values,vec)
df_load=pd.DataFrame(index=pd.MultiIndex.from_tuples(ls_edges),columns=['load'],data=load)
df_load.index.levels[0].name='Ori'
```

e) Compute the external (edge) link loads.

In [63]:
```
df_ext_load=tmo.get_ext_load(df_trafic,ls_nodes)
df_ext_load.index.names=['node','I/O']
#df_ext_load.index.levels[1].name='I/O'
df_ext_load.xs('out', level=df_ext_load.index.levels[1].name)
```

Out[63]:

External	load
node	
A	98
B	186
C	108
D	123
E	84

In [64]:
```
df_ext_load
```

Out[64]:

	External	load
node	**I/O**	
A	**in**	93
	out	98
B	**in**	118
	out	186
C	**in**	158
	out	108
D	**in**	81
	out	123
E	**in**	149
	out	84

f) Compute the gravity model from the traffic matrix from the traffic at the edge links,and compare it with the original traffic matrix.

In [65]:
```
df_grav=tmo.gravity_model(df_ext_load,ls_nodes)
df_grav.convert_objects()
```

Out[65]:

To	A	B	C	D	E
From					
A	15.215359	28.878130	16.767947	19.096828	13.041736
B	19.305509	36.641068	21.275459	24.230384	16.547579
C	25.849750	49.061770	28.487479	32.444073	22.156928
D	13.252087	25.151920	14.604341	16.632721	11.358932
E	24.377295	46.267112	26.864775	30.595993	20.894825

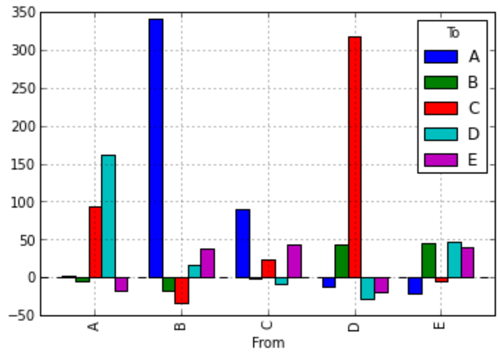
In [66]:
```
import numpy as np
error=tmo.vectorize(df_grav.values,flat=True)-tmo.vectorize(df_trafic.values,flat=True)
esq=error**2
mse=np.sum(esq)/len(df_grav.values)
RMSE=np.sqrt(mse)
print "RMSE del gravity model: ",RMSE
```

```
RMSE del gravity model:  19.62303794
```

A continuación mostramos el error porcentual (expresada en 100%) en cada uno de los valores del gravity model respecto de la matriz real.

In [67]:
```
df_pctg=(df_tomo/df_trafic-1)*100

df_pctg.convert_objects().plot(kind='bar')
```

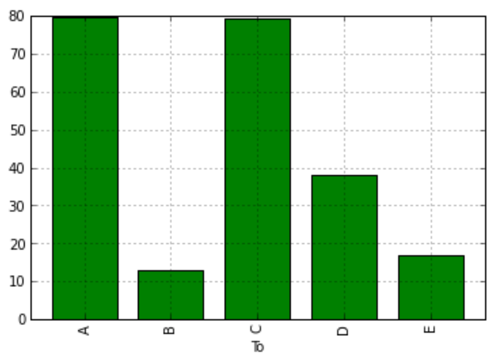
Out[67]:
```
<matplotlib.axes.AxesSubplot at 0x8716390>
```



A continuación mostramos la media y la desviación típica de los dinstintos errores cometidos

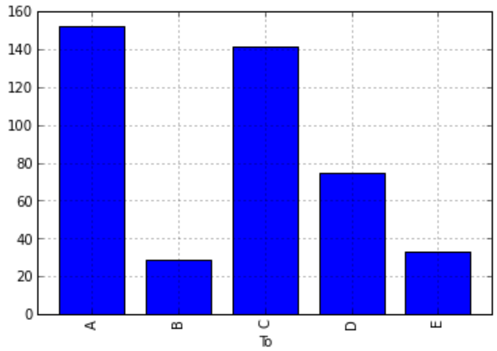
In [68]:
```
#media
df_stat=df_pcte.mean()
df_stat.plot(kind='bar', color='g')
```

Out[68]:
```
<matplotlib.axes.AxesSubplot at 0x8595550>
```



In [69]:
```
#desviación típica
df_stat=df_pcte.std()
df_stat.plot(kind='bar', color='b')
```

Out[69]: <matplotlib.axes.AxesSubplot at 0x85a4810>



g) With Matlab, and from the link load vector, the routing matrix, and the gravity model, compute the tomogravity estimation of the traffic matrix, and compare it with the original TM. Discuss the results.

In [70]:
```
def weight(x):
    return 1/np.sqrt(x)
df_weight=df_grav.applymap(weight)
df_tomo=tmo.tomogravity(df_route,df_load,df_grav,df_weight,ls_nodes)
df_tomo
```

Out[70]:

To	A	B	C	D	E
From					
A	15.21536	32.47401	23.22101	20.86985	19.75723
B	17.60395	36.64107	22.81491	28.12686	16.6688
C	30.41765	53.44483	28.48748	41.2013	28.7839
D	17.57808	28.53768	20.90567	16.63272	10.37594
E	33.94762	49.25052	32.14998	33.97437	20.89482

Estudio del error del modelo

In [71]:
```
error=tmo.vectorize(df_tomo.values,flat=True)-tmo.vectorize(df_trafic.values,flat=True)
esq=error**2
mse=np.sum(esq)/len(df_grav.values)
RMSE=np.sqrt(mse)
print "RMSE del tomogravity model: ",RMSE
```

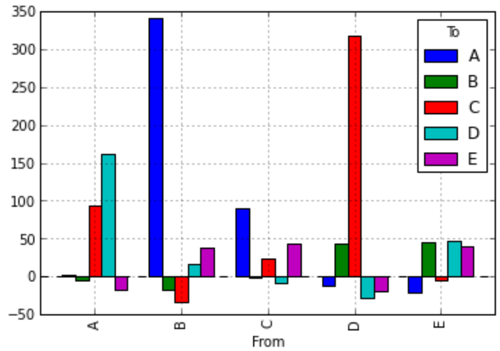
```
RMSE del tomogravity model:  19.4795311206
```

Calculamos un dataframe que contiene el error del modelo en porcentaje para cada valor de la matriz y graficamos sus valores:

In [72]:
```
df_pcte=(df_tomo/df_trafic-1)*100
df_pcte.convert_objects().plot(kind='bar')
df_pcte
```

Out[72]:

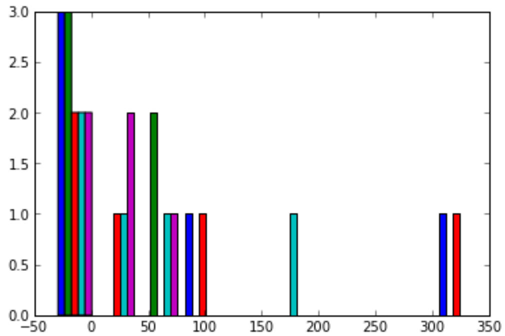
To	A	B	C	D	E
From					
A	1.435726	-4.488211	93.50838	160.8731	-17.67819
B	340.0986	-16.72484	-32.89733	17.19525	38.90671
C	90.11033	-1.02809	23.8586	-8.441549	43.91948
D	-12.1096	42.68839	318.1134	-27.68382	-20.18508
E	-21.05205	44.85447	-5.441225	47.71464	39.29883



Aquípodemos observar un histograma del porcentaje de error cometido respecto del valor real

In [73]:
```
plt.hist(df_pcte.values)
```

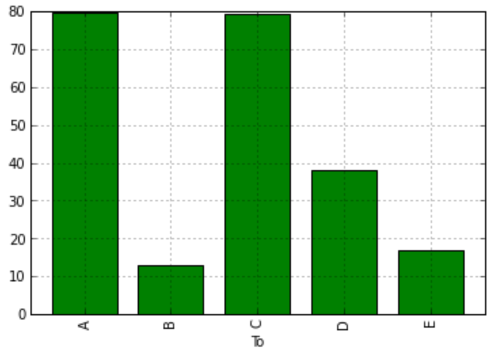
Out[73]:
```
([array([3, 0, 0, 1, 0, 0, 0, 0, 0, 1]),
  array([3, 0, 2, 0, 0, 0, 0, 0, 0, 0]),
  array([2, 1, 0, 1, 0, 0, 0, 0, 0, 1]),
  array([2, 1, 1, 0, 0, 1, 0, 0, 0, 0]),
  array([2, 2, 1, 0, 0, 0, 0, 0, 0, 0])],
 array([ -32.89732638,    4.40227034,   41.70186706,   79.00146378,
        116.3010605 ,  153.60065721,  190.90025393,  228.19985065,
        265.49944737,  302.79904409,  340.0986408 ]),
 <a list of 5 Lists of Patches objects>)
```



A continuación mostramos la media y la desviación típica de los dinstintos errores cometidos

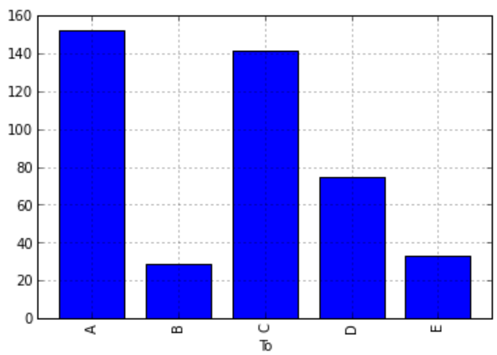
In [74]:
```
#media
df_stat=df_pcte.mean()
df_stat.plot(kind='bar', color='g')
```

Out[74]: <matplotlib.axes.AxesSubplot at 0x9052810>



In [75]:
```
#desviación típica
df_stat=df_pcte.std()
df_stat.plot(kind='bar', color='b')
```

Out[75]: <matplotlib.axes.AxesSubplot at 0x8b84b90>

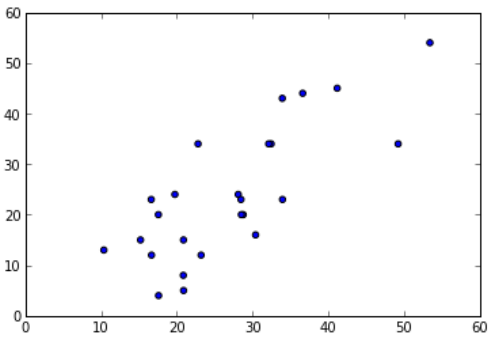


A continuación mostramos un gráfico que nos relaciona las prediciones de nuestro modelo(eje vertical) con los valores reales de la matriz de tráfico (eje horizontal).

En un modelo perfecto el resultado debería ser una recta de pendiente 1 que pasara por el origen. Una mayor desviación respecto de esta figura significará una disminución de la fiabilidad de nuestro modelo. Aun así, en este caso es posible observar una dispersión relativamente lineal que nos indica que nuestro modelo pese a contener errores sería válido para una primera estimación de la matriz de tráfico.

In [76]:
```
plt.scatter(df_tomo,df_trafic)
```

Out[76]: <matplotlib.collections.PathCollection at 0x92ff290>



Después de realizar este estudio no puedo afirmar con rotundidad que los calculos realizados sean correctos, pero aun en el caso de haber implementado mal el modelo creo que es posible utilizarlo como una aproximación razonable para la matriz de tráfico.

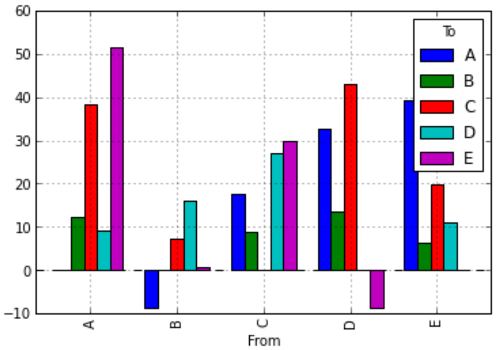
Comparación cruzada de modelos

Para llevar a cabo este estudio empezaremos realizando una comparación porcentual de las diferencias entre los dos modelos.

In [77]:
```
df_modc=(df_tomo/df_grav-1)*100

df_modc.convert_objects().plot(kind='bar')
df_modc
```

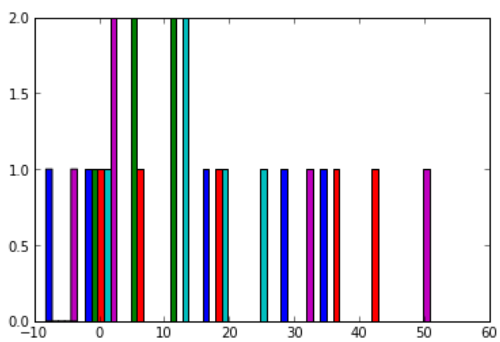
Out[77]:

To	A	B	C	D	E
From					
A	0	12.45191	38.48449	9.28438	51.49236
B	-8.813876	0	7.235801	16.08095	0.7325875
C	17.67097	8.933762	0	26.99177	29.90923
D	32.64386	13.46123	43.14695	0	-8.653907
E	39.25917	6.448229	19.67338	11.04189	0



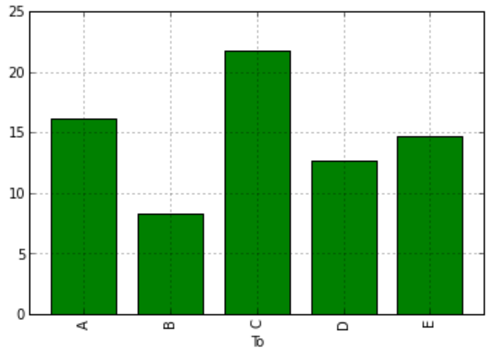
In [78]:
```
plt.hist(df_modc.values)
```

Out[78]:
```
([array([1, 1, 0, 0, 1, 0, 1, 1, 0, 0]),
  array([0, 1, 2, 2, 0, 0, 0, 0, 0, 0]),
  array([0, 1, 1, 0, 1, 0, 0, 1, 1, 0]),
  array([0, 1, 0, 2, 1, 1, 0, 0, 0, 0]),
  array([1, 2, 0, 0, 0, 0, 1, 0, 0, 1])],
 array([ -8.81387553,  -2.78325193,   3.24737167,   9.27799527,
        15.30861887,  21.33924247,  27.36986607,  33.40048967,
        39.43111327,  45.46173687,  51.49236047]),
 <a list of 5 Lists of Patches objects>)
```



In [79]:
```python
#media
df_stat=df_modc.mean()
df_stat.plot(kind='bar', color='g')
```

Out[79]: <matplotlib.axes.AxesSubplot at 0x972f4d0>



A continuación realizaremos un scatter plot entre las predicciones de ambos modelos para estudiar como difieren.

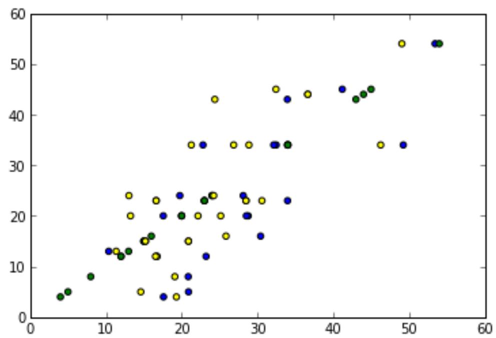
In [80]:
```python
plt.scatter(df_grav,df_tomo) si
```

```
  File "<ipython-input-80-8366b8812b09>", line 1
    plt.scatter(df_grav,df_tomo) si
                                  ^
SyntaxError: invalid syntax
```

Es posible observar un comportamiento fuertemente lineal pero con una pendiente cercana a 1. Esto se podria interpretar como una diferencia sistemática en las predicciones de ambos modelos. Al estar ambos modelos relacionados es lógico que aparezca un resultado de este tipo.

In [81]:
```python
import matplotlib.pyplot as plt
plt.scatter(df_tomo,df_trafic,c='blue')
plt.scatter(df_trafic,df_trafic,c='green')
plt.scatter(df_grav,df_trafic,c='yellow')
```

Out[81]: <matplotlib.collections.PathCollection at 0x957afd0>



En esta imagen se pueden observar en verde los valores reales de la matriz de tráfico, en azul la estimación del tomogravity y en amarillo la del gravity model.

BONUS TRACK

Como el modelo tomogravity no se si está bien calculado (el código Matlab del paper es cuanto menos confuso a la hora de distribuir los pesos en la matriz pseudoinversa) he decidido crear un modelo propio y analizar como se comporta. Aun no tengo el código listo para exportar como una librería, así que en este caso me limitaré a cargar las predicciones calculadas con anterioridad. A continuación expongo el análisis de los resultado de mi estimación:

In [82]:
```
import cPickle as pickle
pred_2_in = open('pred_2.pkl', 'rb')
pred_2= pickle.load(pred_2_in)
df_custom=pd.DataFrame(index=df_trafic.index,columns=df_trafic.columns,data=pred_2.reshape(5,5))
df_custom
```

Out[82]:

	A	B	C	D	E
A	13.292566	28.207070	17.193743	18.023560	13.965958
B	16.676762	34.723862	20.025260	24.015200	14.491737
C	25.559677	49.450779	26.563100	34.205232	23.086556
D	13.067315	24.784411	15.039347	14.476857	8.897693
E	26.415573	45.921734	26.873855	30.025384	18.633656

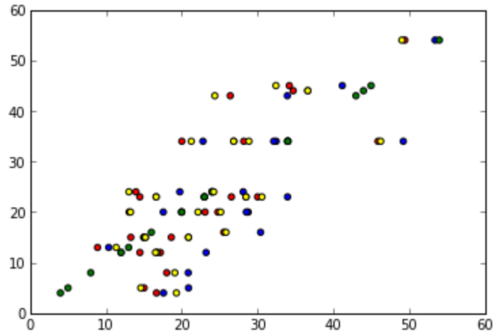
A continuación mostraremos el RMSE del modelo y la comparación de nuestros resultados con los demás modelos. En la imagen se pueden observar en rojo las estimaciones del modelos propio contra las del gravity(amarillo) y las del tomogravity(azul)

In [83]:
```
error=pred_2-tmo.vectorize(df_trafic.values,flat=True)
esq=error**2
mse=np.sum(esq)/len(pred_2)
RMSE=np.sqrt(mse)
print "RMSE de mi modelo: ",RMSE
```

```
RMSE de mi modelo:  8.37952335989
```

In [84]:
```
plt.scatter(df_tomo,df_trafic,c='blue')
plt.scatter(pred_2,df_trafic,c='red')
plt.scatter(df_trafic,df_trafic,c='green')
plt.scatter(df_grav,df_trafic,c='yellow')
#plt.figsize=(20,15)
```

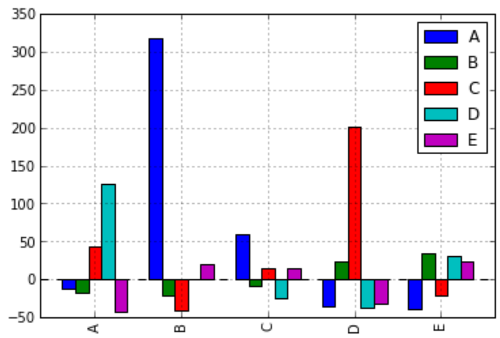
Out[84]: <matplotlib.collections.PathCollection at 0x996ecd0>



In [85]:
```
df_1=df_grav.values-df_trafic.values
df_2=df_tomo.values-df_trafic.values
dif=pred_2-tmo.vectorize(df_trafic.values,flat=True)
df_error=pd.DataFrame(index=range(25),columns=['grav','tomo','custom','output'])
df_error['grav']=tmo.vectorize(df_1)
df_error['tomo']=tmo.vectorize(df_2)
df_error['custom']=dif.reshape(25,1)
```

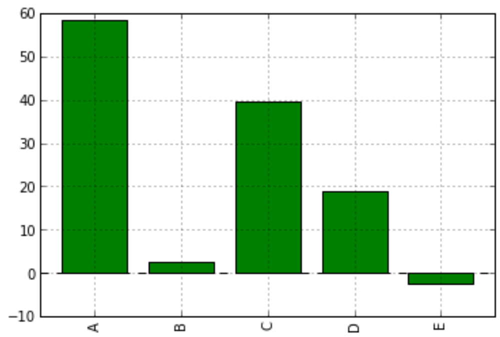
In [86]:
```
df_pctgc=(df_custom/df_trafic-1)*100

df_pctgc.convert_objects().plot(kind='bar')
```

Out[86]: <matplotlib.axes.AxesSubplot at 0x9978310>



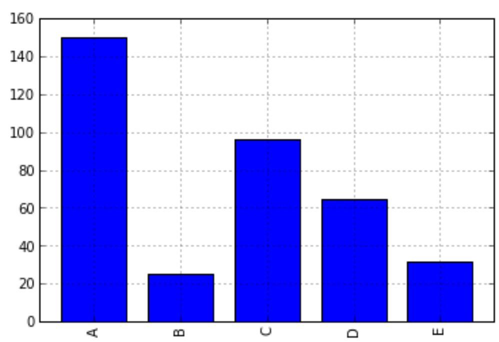
In [87]:
```
#media
df_statc=df_pctgc.mean()

df_statc.plot(kind='bar', color='g')
```

Out[87]: <matplotlib.axes.AxesSubplot at 0x9decd10>



In [88]:
```
#desviación típica
df_statc=df_pctgc.std()
df_statc.plot(kind='bar', color='b')
```

Out[88]: <matplotlib.axes.AxesSubplot at 0x9dff8d0>

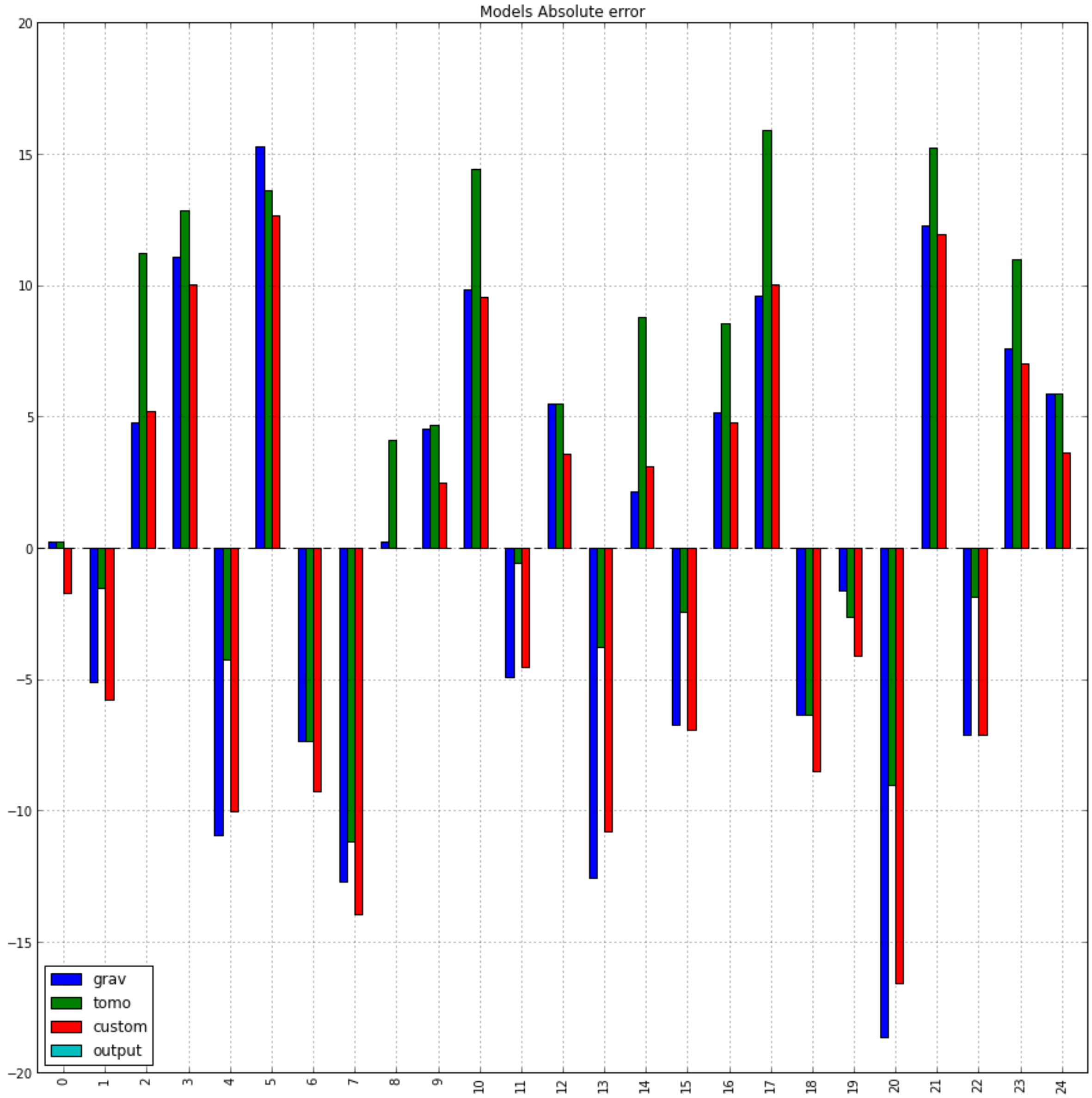


A continuación compararemos los 3 modelos creando dos gráficos de sus errores: El primero mostrando los errores absolutos de cada modelo y el segundo mostrando los errores relativos.

In [88]:

In [89]:
```
df_error.convert_objects().plot(kind='bar',figsize=(15,15),title='Models Absolute error')
```

Out[89]:
```
<matplotlib.axes.AxesSubplot at 0x9f16c50>
```



In [90]:
```
pct_error=df_error.copy()
df_output=pd.DataFrame(index=range(25),columns=['output'],data=tmo.vectorize(df_trafic.values))

pct_error['output']=df_output.values
for col in pct_error.columns:
    pct_error[col]=pct_error[col]/pct_error['output']
del pct_error['output']
```

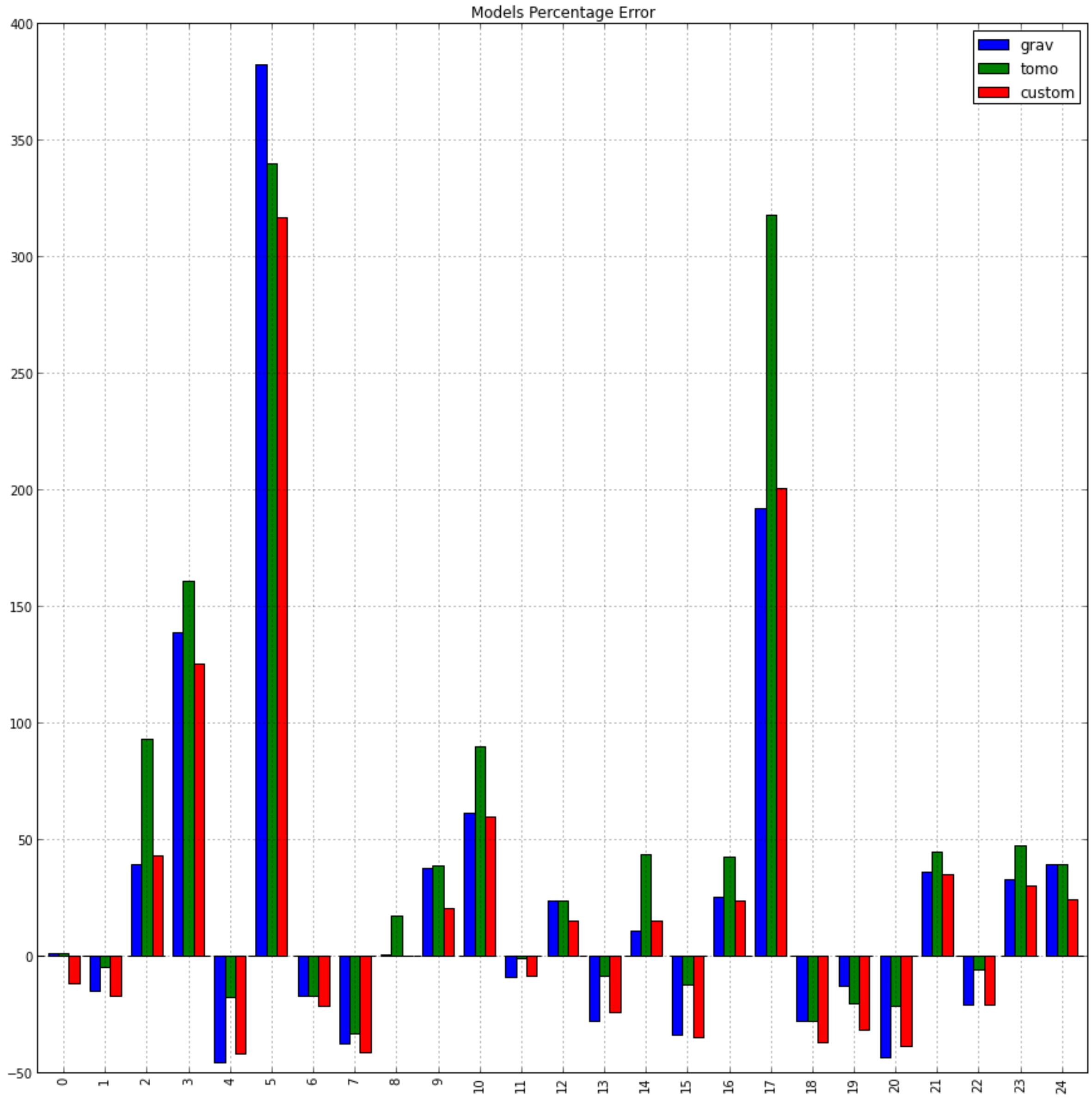
In [91]:
```
pct_error
```

Out[91]:

	grav	tomo	custom
0	0.01435726	0.01435726	-0.113829
1	-0.1506432	-0.04488211	-0.170380
2	0.3973289	0.9350838	0.432812
3	1.387104	1.608731	1.252945
4	-0.4565943	-0.1767819	-0.418085
5	3.826377	3.400986	3.169190
6	-0.1672484	-0.1672484	-0.210821
7	-0.3742512	-0.3289733	-0.411022
8	0.009599332	0.1719525	0.000633
9	0.3789649	0.3890671	0.207645
10	0.6156093	0.9011033	0.597480
11	-0.09144871	-0.0102809	-0.084245
12	0.238586	0.238586	0.154917
13	-0.2790206	-0.08441549	-0.239884
14	0.1078464	0.4391948	0.154328
15	-0.3373957	-0.121096	-0.346634
16	0.257596	0.4268839	0.239221
17	1.920868	3.181134	2.007869
18	-0.2768382	-0.2768382	-0.370571
19	-0.126236	-0.2018508	-0.315562
20	-0.4330862	-0.2105205	-0.385684
21	0.3607974	0.4485447	0.350639
22	-0.2098596	-0.05441225	-0.209592
23	0.3302606	0.4771464	0.305451
24	0.3929883	0.3929883	0.242244

In [92]:
```
(pct_error*100).convert_objects().plot(kind='bar',figsize=(15,15),title='Models Percentage Error')
```

Out[92]: <matplotlib.axes.AxesSubplot at 0xa5ee8d0>



In [92]:

Chunk of code

r'A·X'

In [29]:
```
df_pcte.std()
```

Out[29]:
```
To
A     152.168576
B      28.645598
C     141.489114
D      74.323912
E      32.737584
dtype: float64
```

In [2]:
```
import networkx as nx
import df_network as dfn
import custom_net as cn
import traffic_matrix_operators as tmo
```

In [3]:
```
from IPython.display import Math
import numpy as np
import pandas as pd
#import matplotlib as plt
import networkx as nx
import math as mth
import pylab
import networkx as nx
import matplotlib.pyplot as plt
```

In [4]:
```
def create_colors(G):
    colors=[]
    for a,b in G.edges():
        if G[a][b]['weight'] == 1:
          colors.append((0.0, 0.0, 0.5, 1.0))
        if G[a][b]['weight'] == 2:
          colors.append((0.0, 0.42549019607843136, 1.0, 1.0))
        if G[a][b]['weight'] == 3:
          colors.append((0.3636938646426312, 1.0, 0.60404807084123968, 1.0))
        if G[a][b]['weight'] == 4:
          colors.append((0.85705249841872222, 1.0, 0.11068943706514867, 1.0))
        if G[a][b]['weight'] == 5:
          colors.append((0.5, 0.0, 0.0, 1.0))
    return colors

def vectorize(rm):

    return rm.reshape((size(rm), 1))
def print_link_legends():
     from termcolor import colored

     print colored('5:', 'red'), 'Red'
     print colored('4:', 'yellow'), 'Yellow'
     print colored('3:', 'green'),'Green'
     print colored('2', 'blue'), 'Light Blue'
     print ('1'),'Dark Blue'
G,ls_nodes,ls_edges=tmo.init_network()

labels={}
for i in range(1,len(ls_nodes)+1):
    labels[i]=i
edgecolors=create_colors(G)
```

In [5]:
```
tmatrix=np.array([[15,34,12,8,24],[4,44,34,24,12],[16,54,23,45,20],[20,20,5,23,13],[43,34,34,23,15]])
df_trafic=pd.DataFrame(index=ls_nodes,columns=ls_nodes,data=tmatrix)
df_trafic
```

Out[5]:

	A	B	C	D	E
A	15	34	12	8	24
B	4	44	34	24	12
C	16	54	23	45	20
D	20	20	5	23	13
E	43	34	34	23	15

In [ ]: