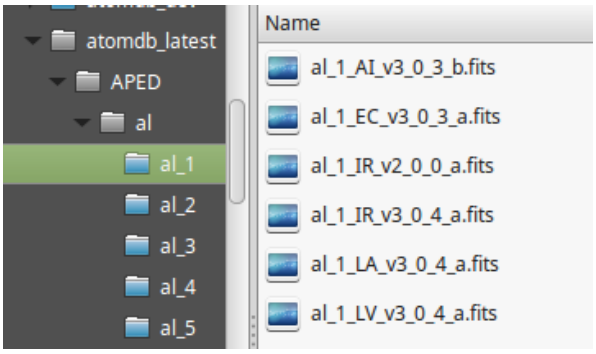
PyAtomDB - what is it?</h1>

PyAtomDB interacts with the AtomDB atomic database. So first, a quick description of AtomDB. AtomDB is primarily targeted at modeling X-ray emission from collisionally ionized astrophysical plasma. (10^4 < T < 10^9 K). It is a combination of several components:

- An Atomic Database (APED)
- An "Plasma Code" (APEC)
- Other assorted models
- Connections to spectral analysis tools

The APED Atomic Database

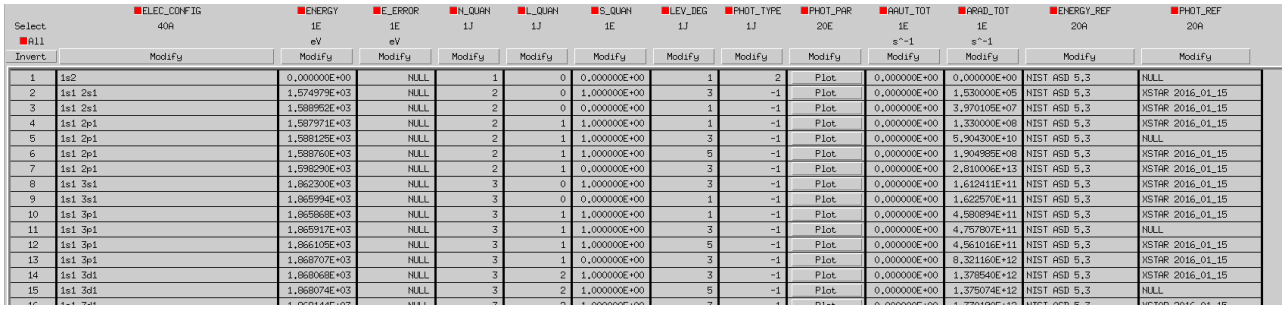
- large collection of FITS files covering different atomic data for all ions from H I to Ni XXVIII, and some more up to Zn.



Different files for different data types

- IR: Ionization and Recomination Rates
- LV: Energy Levels
- LA: Wavelengths and Transition Probabilities
- EC: Electron Collisions
- PC: Proton Collisions
- DR: Dielectronic Recombination Satellite Lines
- AI: Autoionization Rates
- PI: Photoionization Cross Sections

Plus additional files for Abundances



	ELEC_CONFIG	ENERGY	E_ERROR	N_QUAN	L_QUAN	S_QUAN	LEV_DEG	PHOT_TYPE	PHOT_PAR	AAUT_TOT	ARAD_TOT	ENERGY_REF	PHOT_REF
Select	40A	1E	1E	1J	1J	1E	1J	1J	20E	1E	1E	20A	20A
All		eV	eV							s^-1	s^-1		
Invert	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify	Modify
1	1s2	0.000000E+00	NULL	1	0	0.000000E+00	1	2	Plot	0.000000E+00	0.000000E+00	NIST ASD 5.3	NULL
2	1s1 2s1	1.574979E+03	NULL	2	0	1.000000E+00	3	-1	Plot	0.000000E+00	1.530000E+05	NIST ASD 5.3	XSTAR 2016_01_15
3	1s1 2s1	1.588952E+03	NULL	2	0	0.000000E+00	1	-1	Plot	0.000000E+00	3.970105E+07	NIST ASD 5.3	XSTAR 2016_01_15
4	1s1 2p1	1.587971E+03	NULL	2	1	1.000000E+00	1	-1	Plot	0.000000E+00	1.330000E+08	NIST ASD 5.3	XSTAR 2016_01_15
5	1s1 2p1	1.588125E+03	NULL	2	1	1.000000E+00	3	-1	Plot	0.000000E+00	5.904300E+10	NIST ASD 5.3	NULL
6	1s1 2p1	1.588760E+03	NULL	2	1	1.000000E+00	5	-1	Plot	0.000000E+00	1.904985E+08	NIST ASD 5.3	XSTAR 2016_01_15
7	1s1 2p1	1.598290E+03	NULL	2	1	0.000000E+00	3	-1	Plot	0.000000E+00	2.810006E+13	NIST ASD 5.3	XSTAR 2016_01_15
8	1s1 3s1	1.862300E+03	NULL	3	0	1.000000E+00	3	-1	Plot	0.000000E+00	1.612411E+11	NIST ASD 5.3	XSTAR 2016_01_15
9	1s1 3s1	1.865994E+03	NULL	3	0	0.000000E+00	1	-1	Plot	0.000000E+00	1.622570E+11	NIST ASD 5.3	XSTAR 2016_01_15
10	1s1 3p1	1.865868E+03	NULL	3	1	1.000000E+00	1	-1	Plot	0.000000E+00	4.580894E+11	NIST ASD 5.3	XSTAR 2016_01_15
11	1s1 3p1	1.865917E+03	NULL	3	1	1.000000E+00	3	-1	Plot	0.000000E+00	4.757807E+11	NIST ASD 5.3	NULL
12	1s1 3p1	1.866105E+03	NULL	3	1	1.000000E+00	5	-1	Plot	0.000000E+00	4.561016E+11	NIST ASD 5.3	XSTAR 2016_01_15
13	1s1 3p1	1.868707E+03	NULL	3	1	0.000000E+00	3	-1	Plot	0.000000E+00	8.321160E+12	NIST ASD 5.3	XSTAR 2016_01_15
14	1s1 3d1	1.868068E+03	NULL	3	2	1.000000E+00	3	-1	Plot	0.000000E+00	1.378540E+12	NIST ASD 5.3	XSTAR 2016_01_15
15	1s1 3d1	1.868074E+03	NULL	3	2	1.000000E+00	5	-1	Plot	0.000000E+00	1.375074E+12	NIST ASD 5.3	NULL

Online Access: Webguide

There is limited online access to the database to find energies, levels, transitions, etc. See here: [AtomDB Webguide] (http://www.atomdb.org/Webguide)

PyAtomDB

One stop shop for all AtomDB things:

- Access to atomic database
- All model code for turning database into emissivities
- Generating spectra

Installation

If you have pip installed:

```
pip install pyatomdb
```

Note, that if you don't have admin permissions, adding --user will install in your local tree

```
pip install pyatomdb --user
```

Alternatively, the Github Version (https://github.com/AtomDB/pyatomdb) updates more frequently. clone, unzip then run `python setup.py install` or `python setup.py install --user` as appropriate.

As of November 21st 2018, python 2.7 support has been suspended. There is still a GitHub branch for this if required, but Python 3 is now the defaul version.

Initialization

One installed, you must initalize the database by calling from the python interpreter:

```
import pyatomdb
pyatomdb.util.initialize()
```

This should install the files if you follow the on screen instructions. You only need to do this once, not each time you restart pyatomdb. Code below will do this for you.

In [ ]:
```
# NOTE ONLY RUN THIS IF YOU HAVE TO:
# If this is your first installation of pyatomdb
# It will download ~0.5GB of data files
# (It won't break anything to do it again, but you will have an extra 500MB of
downloads to do...)

import pyatomdb
pyatomdb.util.initialize()
```

In [1]:
```
import pyatomdb, numpy
# get the energy level data. Will download files on demand if required.
a = pyatomdb.atomdb.get_data(8, 7, 'LV')

# data is in FITS files, so all the data is in HDU 1 (a[1].data, in this case)
print(a[1].data.dtype)
print('\n\n')
# print details of the first 5 levels
print("Config    Energy (eV)  S  L g")
for aa in a[1].data[:5]:
    print("%-10s %10.4f %.1f %1i %1i"%(aa['ELEC_CONFIG'], aa['ENERGY'], aa['S_QUAN'],aa['L_QUAN'], aa['LEV_DEG']))
```

```
(numpy.record, [('ELEC_CONFIG', 'S40'), ('ENERGY', '>f4'), ('E_ERROR', '>f4'), ('N_QUAN', '>i4'), ('L_QUAN', '>i4'), ('S_QUAN', '>f4'), ('LEV_DEG', '>i4'), ('PHOT_TYPE', '>i4'), ('PHOT_PAR', '>f4', (20,)), ('AAUT_TOT', '>f4'), ('ARAD_TOT', '>f4'), ('ENERGY_REF', 'S20'), ('PHOT_REF', 'S20'), ('AAUT_REF', 'S20'), ('ARAD_REF', 'S20')])



Config    Energy (eV)  S  L g
1s2            0.0000 0.0 0 1
1s1 2s1      560.9839 1.0 0 3
1s1 2s1      568.8866 0.0 0 1
1s1 2p1      568.5444 1.0 1 1
1s1 2p1      568.5519 1.0 1 3
```

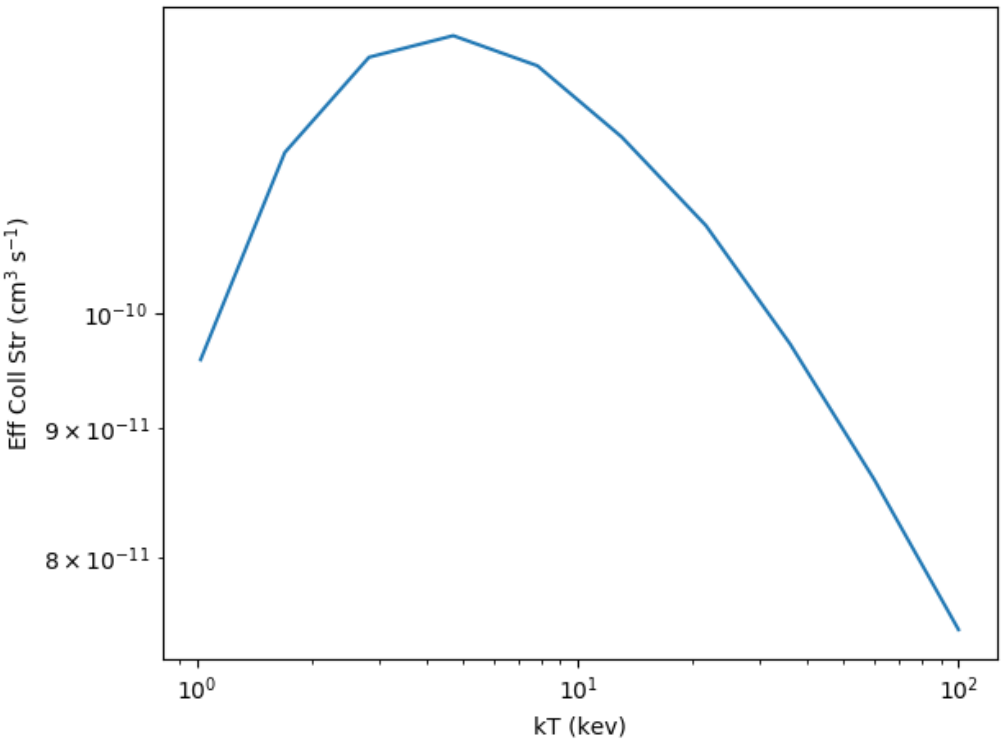
Accessing the Atomic Database

Let's get some data on He-like oxygen

In [2]:
```
# now lets find an effective collision strength
Te = numpy.logspace(0.01,2.0,10) # keV
initlev = 1
finallev = 7
Z=8
z1 = 7
dtype='EC'
exc, dex = pyatomdb.atomdb.get_maxwell_rate(Te, dtype=dtype, initlev = initlev, \
                                            finallev=finallev, Te_unit='keV', \
Z=Z, z1=z1 )
```

```
ladat! False
```

In [3]:
```
# some plotting setup
%matplotlib notebook
import matplotlib.pyplot as plt
plt.loglog(Te, exc)
plt.xlabel('kT (kev)')
plt.ylabel('Eff Coll Str (cm$^3$ s$^{-1}$)')
plt.tight_layout()
plt.show()
```



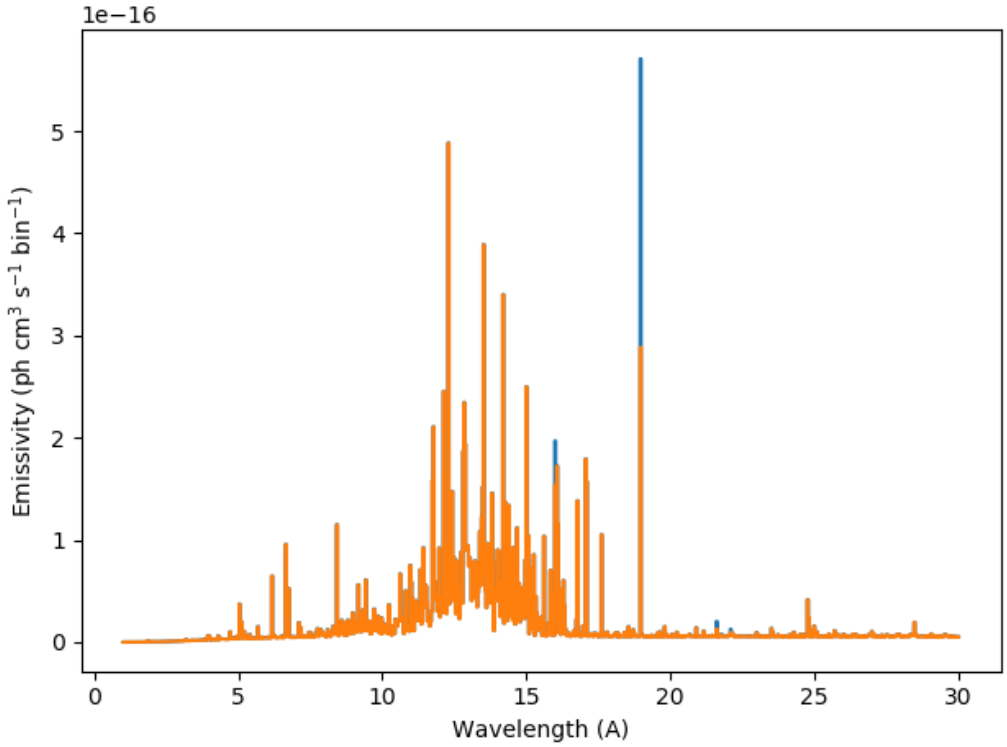
Making Spectra

Calculate a spectrum and fold it through an instrument response

$$\epsilon(T_e) = N_k A_{kj}$$

$$N_k = \frac{N_k}{N_z}\frac{N_z}{N_Z}\frac{N_Z}{N_{Hyd}}\frac{N_{Hyd}}{N_e}N_e$$

In [4]:
```
s = pyatomdb.spectrum.Session()
ebins = numpy.linspace(1,30,901)
s.set_specbins(ebins, specunits='A')

# get the spectrum at kT=1keV

spec1 = s.return_spectra(1.0)

# plotting

plt.figure()
plt.plot(ebins, numpy.append(0, spec1), drawstyle='steps')
plt.xlabel('Wavelength (A)')
plt.ylabel('Emissivity (ph cm$^3$ s$^{-1}$ bin$^{-1}$)')
plt.tight_layout()
plt.show()
```

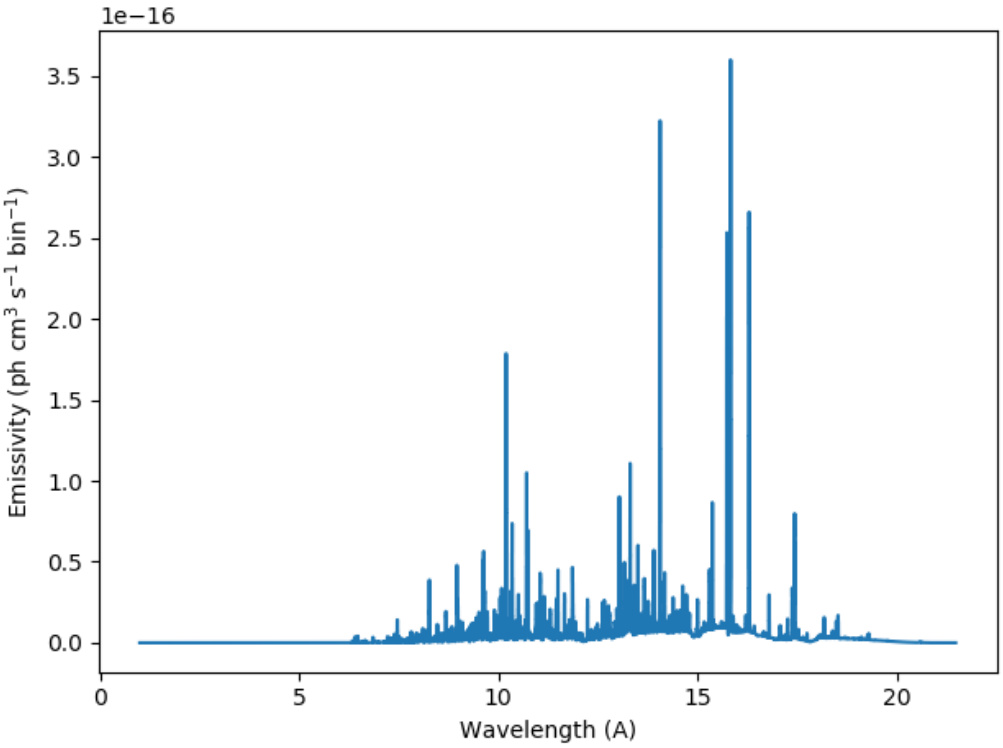


In [5]:
```
# Halve the oxygen
s.set_abund(8,0.5)
spec2 = s.return_spectra(1.0)
plt.plot(ebins, numpy.append(0, spec2), drawstyle='steps')
plt.show()
```

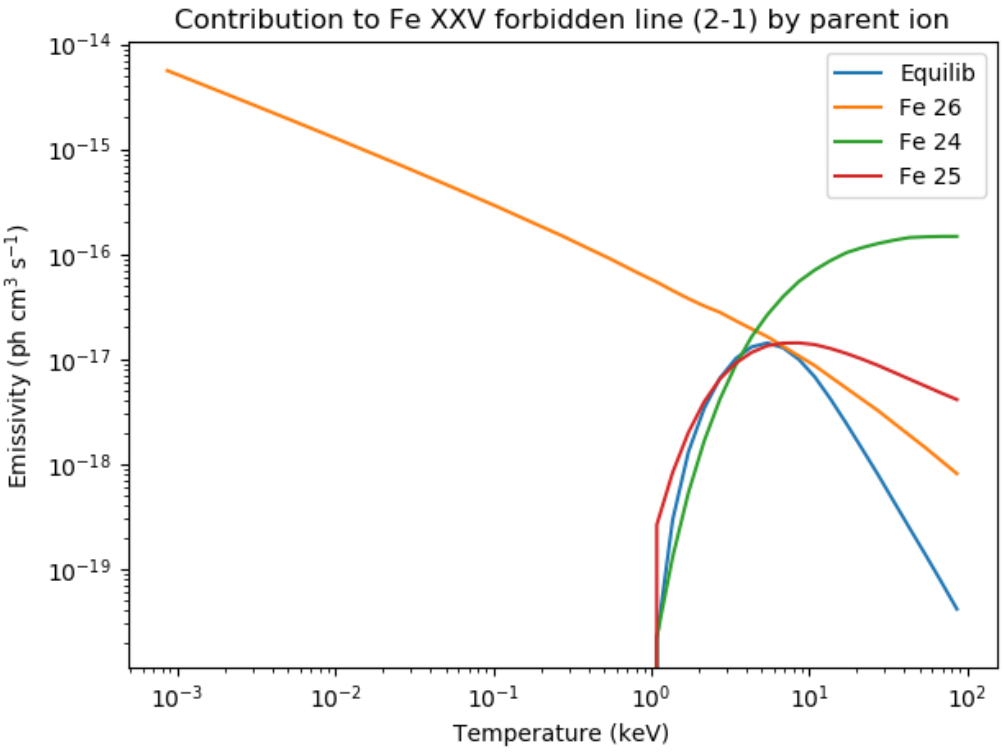
In [6]:
```
s2 = pyatomdb.spectrum.Session()
s2.set_response('aciss_heg1_cy19.grmf', arf='aciss_heg1_cy19.garf')
spec3 = s2.return_spectra(1.0)
# NOTE THIS TAKES ~ 1 minute
```

```
starting apply response at Mon Jun  3 12:02:38 2019
starting apply response at Mon Jun  3 12:02:39 2019
starting apply response at Mon Jun  3 12:02:39 2019
starting apply response at Mon Jun  3 12:02:40 2019
starting apply response at Mon Jun  3 12:02:41 2019
starting apply response at Mon Jun  3 12:02:41 2019
starting apply response at Mon Jun  3 12:02:42 2019
starting apply response at Mon Jun  3 12:02:42 2019
starting apply response at Mon Jun  3 12:02:43 2019
starting apply response at Mon Jun  3 12:02:43 2019
starting apply response at Mon Jun  3 12:02:44 2019
starting apply response at Mon Jun  3 12:02:44 2019
starting apply response at Mon Jun  3 12:02:45 2019
starting apply response at Mon Jun  3 12:02:46 2019
starting apply response at Mon Jun  3 12:02:46 2019
starting apply response at Mon Jun  3 12:02:47 2019
starting apply response at Mon Jun  3 12:02:47 2019
starting apply response at Mon Jun  3 12:02:48 2019
starting apply response at Mon Jun  3 12:02:48 2019
starting apply response at Mon Jun  3 12:02:49 2019
starting apply response at Mon Jun  3 12:02:49 2019
starting apply response at Mon Jun  3 12:02:50 2019
starting apply response at Mon Jun  3 12:02:51 2019
starting apply response at Mon Jun  3 12:02:51 2019
starting apply response at Mon Jun  3 12:02:52 2019
starting apply response at Mon Jun  3 12:02:53 2019
starting apply response at Mon Jun  3 12:02:53 2019
starting apply response at Mon Jun  3 12:02:54 2019
starting apply response at Mon Jun  3 12:02:55 2019
starting apply response at Mon Jun  3 12:02:55 2019
starting apply response at Mon Jun  3 12:02:56 2019
starting apply response at Mon Jun  3 12:02:56 2019
starting apply response at Mon Jun  3 12:02:57 2019
starting apply response at Mon Jun  3 12:02:57 2019
starting apply response at Mon Jun  3 12:02:58 2019
starting apply response at Mon Jun  3 12:02:59 2019
starting apply response at Mon Jun  3 12:02:59 2019
starting apply response at Mon Jun  3 12:03:00 2019
starting apply response at Mon Jun  3 12:03:00 2019
starting apply response at Mon Jun  3 12:03:01 2019
starting apply response at Mon Jun  3 12:03:01 2019
starting apply response at Mon Jun  3 12:03:02 2019
starting apply response at Mon Jun  3 12:03:02 2019
starting apply response at Mon Jun  3 12:03:03 2019
starting apply response at Mon Jun  3 12:03:03 2019
starting apply response at Mon Jun  3 12:03:04 2019
starting apply response at Mon Jun  3 12:03:05 2019
starting apply response at Mon Jun  3 12:03:05 2019
starting apply response at Mon Jun  3 12:03:06 2019
starting apply response at Mon Jun  3 12:03:06 2019
starting apply response at Mon Jun  3 12:03:07 2019
starting apply response at Mon Jun  3 12:03:07 2019
starting apply response at Mon Jun  3 12:03:08 2019
starting apply response at Mon Jun  3 12:03:08 2019
starting apply response at Mon Jun  3 12:03:09 2019
starting apply response at Mon Jun  3 12:03:10 2019
starting apply response at Mon Jun  3 12:03:11 2019
starting apply response at Mon Jun  3 12:03:12 2019
starting apply response at Mon Jun  3 12:03:12 2019
starting apply response at Mon Jun  3 12:03:13 2019
```

In [7]:
```
plt.figure()
plt.plot(12.398425/s2.ebins_response[::-1], numpy.append(0, spec3), drawstyle='
steps')
plt.xlabel('Wavelength (A)')
plt.ylabel('Emissivity (ph cm$^3$ s$^{-1}$ bin$^{-1}$)')
plt.tight_layout()
plt.show()
```



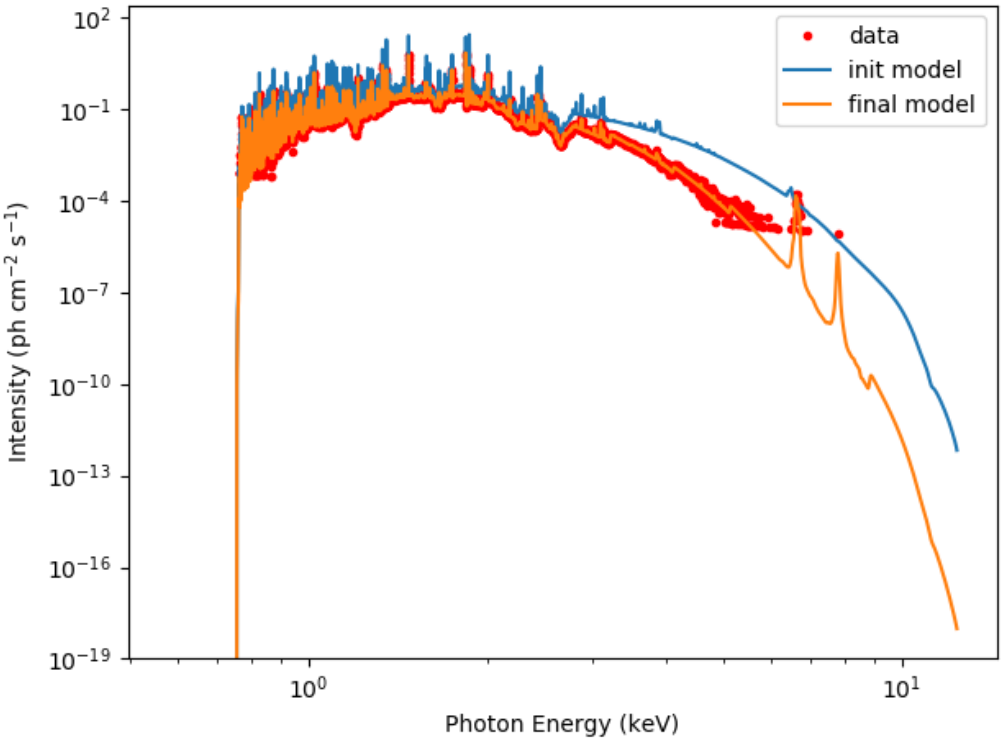
In [8]:
```
#get individual line emissivities vs temperature

emiss = pyatomdb.atomdb.get_line_emissivity(26,25,2,1)
plt.figure()
plt.loglog(emiss['kT'], emiss['epsilon'], label='Equilib')
emiss = pyatomdb.atomdb.get_line_emissivity(26,25,2,1, use_nei_raw=True)
for z1 in emiss['epsilon'].keys():
    plt.loglog(emiss['kT'], emiss['epsilon'][z1], label='Fe %i'%(z1))

plt.xlabel('Temperature (keV)')
plt.ylabel('Emissivity (ph cm$^3$ s$^{-1}$)')
plt.title("Contribution to Fe XXV forbidden line (2-1) by parent ion")
plt.legend(loc=0)

plt.tight_layout()
plt.show()
```



Integration into Modeling Codes

Our codes are written in python, for easy access and development. Most users want only the emissivities, and they want them to run fast in an fitting loop. As such, our emissivity files can be read in by XSPEC for this and other models.

In [9]:
```
import xspec

#spectrum
s1 = xspec.Spectrum('testacx_99.pha')

#model
m1 =xspec.Model('tbabs*vrnei')

#thaw the oxygen content
m1.vrnei.O.frozen=False
m1.vrnei.kT=0.7
m1.vrnei.kT_init=0.7
m1.show()


#set some plot parameters
xspec.Plot.device='/null'
xspec.Plot.xAxis='keV'
xspec.Plot('data')
xspec.Plot.perHz=False
# make the figures

plt.figure()
plt.loglog(xspec.Plot.x(), xspec.Plot.y(), 'r.', label='data')
plt.loglog(xspec.Plot.x(), xspec.Plot.model(), label='init model')

# fit the data
xspec.Fit.perform()


# plot the results
xspec.Plot('data')

plt.loglog(xspec.Plot.x(), xspec.Plot.model(), label='final model')


plt.xlabel('Photon Energy (keV)')
plt.ylabel('Intensity (ph cm$^{-2}$ s$^{-1}$)')
plt.legend(loc=0)
plt.tight_layout()

plt.show()
```

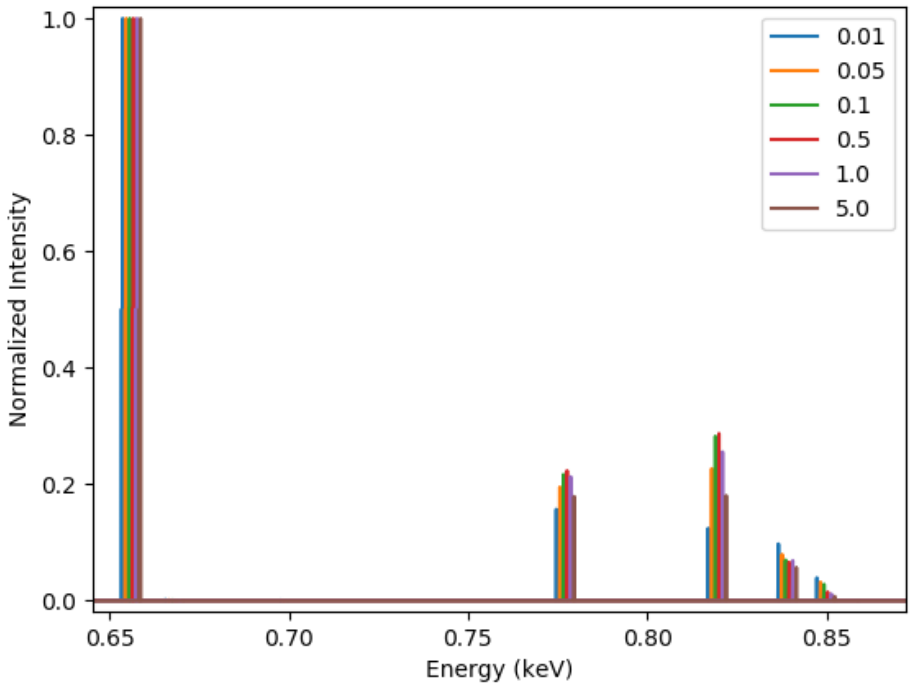


Incorporating our models into XSPEC: ACX

In [10]:
```
import vvacx
xspec.Xset.restore('oxygencx.mdl')
m1=xspec.AllModels(1)
```

```
{{ Model xspecvvacx available to add}}
Specbins shape 8193
Flux shape 8192
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenli_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenbe_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenb_v3.0.7.fi
ts
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenf_v3.0.7.fi
ts
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenna_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenp_v3.0.7.fi
ts
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigencl_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenk_v3.0.7.fi
ts
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigensc_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenti_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenv_v3.0.7.fi
ts
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigencr_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenmn_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenco_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigencu_v3.0.7.f
its
wrote file locally to /export1/atomdb_latest/APED/ionbal/eigen/eigenzn_v3.0.7.f
its
```

In [11]:
```
# adjust velocity to see differences
y={}
celist = [0.01, 0.05, 0.1, 0.5, 1.0, 5.0] #keV/amu
plt.figure()
offset=0.00

for ce in celist:
    m1.xspecvvacx.collenergy=ce
    xspec.Plot('Model')
    y[ce] = xspec.Plot.model()
    plt.plot(numpy.array(xspec.Plot.x())+offset, numpy.array(y[ce])/max(numpy.a
rray(y[ce])), label=repr(ce))
    offset+=0.001
    print(ce, max(numpy.array(y[ce])))
plt.legend(loc=0)
plt.xlabel('Energy (keV)')
plt.ylabel('Normalized Intensity')

plt.show()
```




```
0.01 6920773.964426305
0.05 18989028.742170468
0.1 30652347.638608914
0.5 105849210.21125837
1.0 167820803.9504355
5.0 397240763.4044789
```

In [ ]: